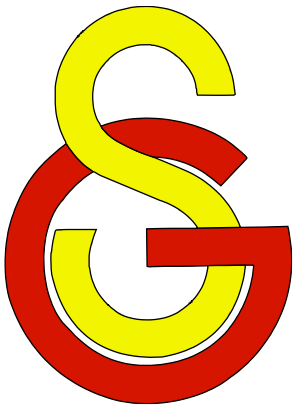
Algorithmique & programmation en langage C - vol.3

Damien Berthet, Vincent Labatut

Université Galatasaray
Faculté d'ingénierie et de technologie

Algorithmique &
programmation en langage C

Damien Berthet & Vincent Labatut



Corrigés de travaux pratiques

Supports de cours – Volume 3
Période 2005-2014


Damien Berthet & Vincent Labatut 2005-2014

Galatasaray Universitesi
Mühendislik ve Teknoloji Fakültesi
Çırağan Cad. No:36
Ortaköy 34349, İstanbul
Turquie

version 1
24/07/2014

Sommaire

Ce recueil est le troisième volume d'une série de trois documents, comprenant également le support de cours (volume 1) et un recueil de sujets de travaux pratiques (volume 2). Ils ont été écrits pour différents enseignements d'algorithmique et de programmation en langage C donnés à la Faculté d'ingénierie de l'Université Galatasaray (Istanbul, Turquie), entre 2005 et 2014. Ce troisième et dernier volume contient les corrigés des 64 sujets de travaux pratiques (TP) et d'examen regroupés dans le deuxième volume.

Malgré tout le soin apporté à la rédaction de ces corrigés, il est probable que des erreurs s'y soient glissées. Merci de nous contacter afin de nous indiquer tout problème détecté dans ce document. Il faut également remarquer qu'il s'agit de TP donnés dans le cadre d'un cours d'introduction, aussi les notions abordées le sont parfois de façon simplifiée et/ou incomplète.

Les conventions utilisées dans ces corrigés sont les mêmes que pour les sujets. Veuillez donc vous référer au deuxième volume pour en connaitre le détail. À noter que la description des outils utilisés, de leur installation et de leur configuration est également traitée dans le deuxième volume.

Damien Berthet & Vincent Labatut
le 19 juillet 2014

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 01 | entrées-sorties

3 Entrées/sorties non-formatées

Exercice 1

```
int main()
{  char c;
   c=getchar();
   putchar(c);
   putchar('\n');

   return EXIT_SUCCESS;
}
```

Exercice 2

```
int main()
{  char c1,c2;
   c1=getchar();   // on saisit le premier caractère
   getchar();      // on consomme le retour chariot restant dans le buffer
   c2=getchar();   // on saisit le second caractère
   putchar(c1);    // on affiche le premier caractère
   putchar('\n');  // on va à la ligne
   putchar(c2);    // on affiche le second caractère
   putchar('\n');  // on va à la ligne

   return EXIT_SUCCESS;
}
```

Au premier appel de `getchar`, on saisit deux caractères :

- le caractère entré par l'utilisateur
- le caractère de fin de ligne `'\n'`

La fonction `getchar` n'utilise que le premier caractère, le second reste dans le buffer. Au deuxième appel de `getchar`, la fonction trouve le caractère `'\n'` dans le tampon, et l'utilise sans attendre que l'utilisateur entre un caractère via le clavier. Il faut un troisième appel de `getchar` pour que l'utilisateur puisse saisir son second caractère.

Il est important de ne pas oublier de vider le tampon quand on utilise des entrées/sorties bufférisées. Il existe deux méthodes :

- consommer les caractères présents dans le tampon, par exemple avec un ou plusieurs `getchar`.
- utiliser la fonction `fflush(stdin)`, qui permet **parfois** de vider le tampon. Parfois seulement, car le comportement de cette commande n'est **pas défini** (dans la norme) pour les flux d'entrée, donc on ne peut pas être **certain** de vider effectivement le tampon.

4 Sorties formatées

Exercice 3

```
int main()
{  double x1=1.2345;
   double x2=123.45;
```

```
    double x3=0.000012345;
    double x4=1e-10;
    double x5=-123.4568e15;
    printf("%%f:%f\t%%e:%e\n",x1,x1);
    printf("%%f:%f\t%%e:%e\n",x2,x2);
    printf("%%f:%f\t%%e:%e\n",x3,x3);
    printf("%%f:%f\t%%e:%e\n",x4,x4);
    printf("%%f:%f\t%%e:%e\n",x5,x5)

    return EXIT_SUCCESS;
}
```

Exercice 4

```
int main()
{  float x1=12.34567f;
   float x2=1.234567;
   float x3=1234567;
   float x4=123456.7;
   float x5=0.1234567;
   float x6=1234.567;
   printf("%15.2f\t%15.2f\n",x1,x2);
   printf("%15.2f\t%15.2f\n",x3,x4);
   printf("%15.2f\t%15.2f\n",x5,x6);

   return EXIT_SUCCESS;
}
```

Exercice 5

```
int main()
{  printf("%%d:%d \n",1234.5678f);
   printf("%%f:%f \n",1234.5678f);
   printf("%%e:%e \n",1234.5678f);

   return EXIT_SUCCESS;
}
```

On observe qu'il est possible d'afficher un réel en tant qu'entier (le contraire est également vrai), sans que le compilateur ne relève d'erreur, et sans provoquer de bug : il n'y a donc ni erreur de compilation, ni erreur d'exécution.

5 Entrées formatées

Exercice 6

```
int main()
{  int x;
   printf("entrez un entier : ");
   scanf("%d",&x);
   printf("le triple de %d est : %d\n",x,3*x);

   return EXIT_SUCCESS;
}
```

Exercice 7

```
int main()
{  int h,m,s;
   printf("entrez l'heure (hh:mm:ss) : ");
   scanf("%d:%d:%d",&h,&m,&s);
   printf("%2d heures\n",h);
   printf("%2d minutes\n",m);
   printf("%2d secondes\n",s);

   return EXIT_SUCCESS;
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 02 | types simples

1 Caractères

Exercice 1

```
int main()
{  unsigned char c;
   // on saisit le caractère
   printf("Entrez un caractere : ");
   c=getchar();

   // on affiche les informations demandées
   printf("Le code ASCII de '%c' est %d\n",c,c);
   printf("Le caractere suivant dans la table ASCII est '%c'\n",c+1);

   return EXIT_SUCCESS;
}
```

Il faut bien faire attention à utiliser une variable de type `unsigned char`, car les codes ASCII sont compris entre 0 et 255, et le type `char` (signé) est limité à des valeurs comprises entre -128 et 127, ce qui pose problème lors de l'affichage du code ASCII. Ainsi, pour le caractère `ü`, on obtient -127 au lieu de 129.

Exercice 2

```
int main()
{  unsigned char c;

   int c;
   printf("Entrez le code ASCII : ");
   scanf("%d",&c);
   printf("Le caractere correspondant au code ASCII %d est '%c'\n",c,c);

   return EXIT_SUCCESS;
}
```

2 Entiers

Exercice 3

```
int main()
{  int v1 = 12 ;
   int v2 = 4294967284;

   printf("entiers naturels : 12=%x et 4294967284=%x\n",v1,v2);
   printf("entiers relatifs : 12=%u et 4294967284=%u\n",v1,v2);
   printf("entiers relatifs : 12=%d et 4294967284=%d\n",v1,v2);

   return EXIT_SUCCESS;
}
```

L'affichage au format hexadécimal donne la décomposition en base 16 des entiers 12 et 4294967284, c'est-à-dire :

- $12 = (0000\ 000c)_{16}$
- $4\ 294\ 967\ 284 = (\text{FFFF FFF3})_{16}$.

On en déduit que ces nombres sont codés par les suites de bits suivantes :

- 12 : **0000 0000 0000 0000 0000 0000 0000 1100**
- 4294967284 : **1111 1111 1111 1111 1111 1111 1111 0011**

Lorsqu'on réalise l'affichage en tant qu'entiers non-signés, les valeurs affichés sont les valeurs binaires des suites de bits, soit les valeurs attendues 12 et 4 294 967 284.

Par contre, l'affichage en tant qu'entiers signés donne 12 et -12. C'est normal, puisque les octets codants 4 294 967 284 sont interprétés comme la valeur en complément à 2 de **1111 1111 1111 1111 1111 1111 1111 0011**, qui est bien -12 car $-12 = 2^{32} - 4\,294\,967\,286$.

Exercice 4

```
int main()
{  printf("valeur +4 : decimal=%2d octal:%11o hexadecimal=%8X\n",4,4,4);
   printf("valeur -4 : decimal=%d octal:%o hexadecimal=%X\n",-4,-4,-4);

   return EXIT_SUCCESS;
}
```

Les affichages de la valeur positive sont normaux. Pour la valeur négative, on obtient des valeurs aberrantes en octal et en hexadécimal, ce qui est normal puisque ces format permettent d'afficher seulement des valeurs non-signées. Ainsi, en hexadécimal on obtient $(\text{FFFFFFFC})_{16}$, soit $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100)_2$. Ce nombre est le complément à deux de $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000\ 0100)_2$, qui correspond bien à la valeur $(4)_{10}$: on a bien une représentation du nombre négatif en complément à 2.

Exercice 5

```
int main()
{  short x,y;

   printf("Entrez la valeur de x : ");
   scanf("%hd",&x);
   printf("Entrez la valeur de y : ");
   scanf("%hd",&y);
   printf("x+y=%hd\n",x+y);

   return EXIT_SUCCESS;
}
```

Pour les calculs proposés, on obtient des valeurs aberrantes, pour cause de dépassement de la capacité du type `short`. Dans le premier cas, on obtient un résultat supérieur à 32767, qui est considéré comme un nombre négatif en raison du codage utilisé pour représenter les entiers signés (complément à deux). Dans le second cas, on obtient un résultat inférieur à -32768, qui est considéré comme un positif pour la même raison.

3 Réels

Exercice 6

La constante `0.1f` est représentée à l'aide de la norme IEEE32. Le développement en base 2 de 0,1 est : $0{,}1 = (0{,}0001\ 1001\ 1001..)_2 = (1{,}1001\ 1001\ 1001\ ....)_2 \times 2^{-4}$.

D'où :

- $s = 0$
- $e + 2^{8-1} - 1 = -5 + 2^7 = (0111\ 1011)_2$
- $m = (1{,}1001\ 1001\ 1001\ ....)_2$

On en déduit le codage de `0.1f` :

0011 1101 1100 1100 1100 1100 1100 1100

Remarque : En réalité, `0.1f` peut être codé en mémoire sous la forme :

`0011 1101 1100 1100 1100 1100 1100 110`**`1`**

Le système effectuant un arrondi pour calculer le dernier bit.

Le programme permettant d'afficher `0.1f` avec 1 et 10 chiffres après la virgule et au format hexadécimal est le suivant :

```
int main()
{  printf("1  decimale  : %.1f\n",0.1f);
   printf("10 decimales : %.10f\n",0.1f);

   return EXIT_SUCCESS;
}
```

Les valeurs affichées sont respectivement `0.1` et `0.1000000015`. Comme il n'existe pas de représentation binaire finie du nombre $(0{,}1)_{10}$, la valeur représentée en mémoire est une approximation.

Remarque : l'écriture d'un nombre réel $x \in [0{,}1[$ en notation binaire s'écrit $x = (0, a_1 \dots a_n)_2$, avec $a_i \in \{0; 1\}$ et $x = a_1 2^{-1} + \cdots + a_n 2^{-n}$. Par exemple : $(0{,}625)_{10}$ s'écrit $(0{,}101)_2$, car :

$$0{,}625 = 0{,}5 + 0{,}125 = \mathbf{1} \times 2^{-1} + \mathbf{0} \times 2^{-2} + \mathbf{1} \times 2^{-3}$$

Un nombre réel *x* possède une écriture binaire finie *si et seulement si* il existe un entier *k* positif tel que $2^k x$ est un entier. Or, quel que soit $k \in \mathbb{N}$, $2^k/10$ n'est pas entier. Donc le nombre $(0{,}1)_{10}$ ne possède pas de développement fini en écriture binaire.

Exercice 7

Le programme est :

```
int main()
{  printf("(1e-9+1e9)-1e9 : %.10f\n",(1e-9+1e9)-1e9);
   printf("1e-9+(1e9-1e9) : %.10f\n",1e-9+(1e9-1e9));

   return EXIT_SUCCESS;
}
```

L'affichage obtenu est :

```
(1e-9+1e9)-1e9 : 0.0000000000
1e-9+(1e9-1e9) : 0.0000000010
```

Conclusion : l'**addition** des nombres à virgule flottante n'est **pas commutative** !

> *"It makes me nervous to fly an airplane since I know they are designed using floating-point arithmetic."* A. Householder
>
> (Un des fondateurs de l'arithmétique numérique)

Le résultat de la première opération est inexact, en effet `1e-9+1e9` ne peut être codé de manière exacte dans le type `float` car ce nombre a trop de chiffres significatifs.

Pour obtenir le résultat attendu, on peut augmenter l'espace mémoire alloué au codage de ces constantes en les déclarants de type `long`. Ainsi, ces constantes seront codées sur 8 octets au lieu de 4 pour le type `float` ce qui augmentera le nombre de chiffres significatifs du résultat.

L'exécution du code suivant donne le résultat attendu, c'est-à-dire 0 :

```
   printf("(1e-9+1e9)-1e9 : %.10f\n",(1e-9l+1e9l)-1e9l);
   printf("1e-9+(1e9-1e9) : %.10f\n",1e-9l+(1e9l-1e9l));
```

Remarque : on a suffixé les constantes avec `l` pour qu'elles soient codées dans le type `long`.

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 03 | variables & opérateurs

1 Variables & adresses

Exercice 1

```
short a,b,c;

int main()
{   short d,e,f;
    printf("&a:%p &b:%p &c:%p \n",&a,&b,&c);
    printf("&d:%p &e:%p &f:%p \n",&d,&e,&f);

    return EXIT_SUCCESS;
}
```

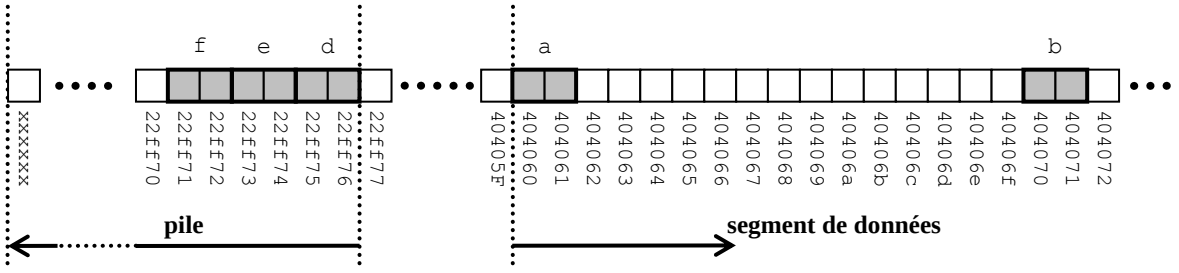
On observe que les variables globales et locales occupent des endroits différents de la mémoire. Par exemple :

```
&a:0x404060 &b:0x404070 &c:0x404080
&d:0x22ff76 &e:0x22ff74 &f:0x22ff72
```

Ici, les adresses des variables stockées dans le segment de données (variables globales) commencent par `0x40`, alors que celles des variables stockées dans la pile (variables locales) commencent par `0x22`.

De plus, on s'aperçoit que les adresses sont affectées aux variables différemment :

- Pour les variables globales :
  - $(10)_{16} = 16$ octets séparent chaque variable, alors que les variables n'occupent que *2* octets. **Remarque :** ceci n'est pas systématique, l'espace laissé entre les variables dépend de la machine.
  - une nouvelle variable est stockée **après** la dernière variable crée
- Pour les variables locales :
  - 2 octets séparent chaque variable, ce qui correspond bien à la place occupée par les variables de type `short`.
  - une nouvelle variable est stockée **avant** la dernière variable crée



Exercice 2

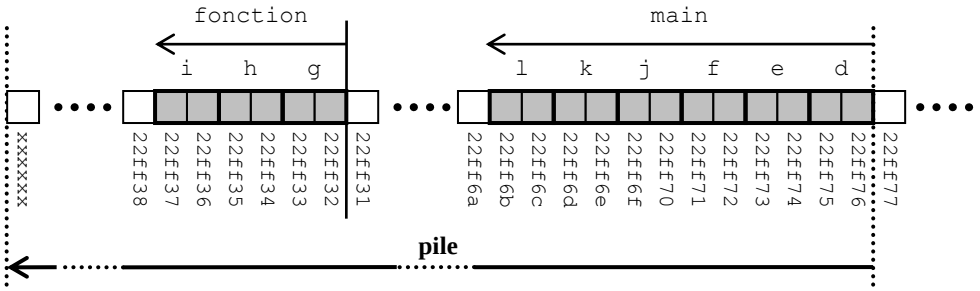
```
short a,b,c;

void fonction()
{   short g,h,i;
    printf("&g:%p &h:%p &i:%p \n",&g,&h,&i);
}
```

```
int main()
{  short d,e,f;
   printf("&a:%p &b:%p &c:%p \n",&a,&b,&c);
   printf("&d:%p &e:%p &f:%p \n",&d,&e,&f);

   fonction();

   short j,k,l;
   printf("&j:%p &k:%p &l:%p \n",&j,&k,&l);

   return EXIT_SUCCESS;
}
```

On observe que les variables crées dans `fonction` n'occupent pas un espace mémoire contigu aux variables crées dans le `main`. Par exemple :

```
&a:0x404060 &b:0x404070 &c:0x404080
&d:0x22ff76 &e:0x22ff74 &f:0x22ff72
&g:0x22ff36 &h:0x22ff34 &i:0x22ff32
&j:0x22ff70 &k:0x22ff6e &l:0x22ff6c
```

En fait, les variables de `fonction` sont dans la pile, mais à un endroit différent. De plus, quand `fonction` est terminée, on peut constater que les nouvelles variables (`j`, `k` et `l`) déclarées dans le `main` sont stockées à des adresses contigües à celles des variables précédemment crées dans le `main` (`d`, `e` et `f`).



2 Opérateurs & transtypage

Exercice 3

```
int main()
{  char c;
   printf("Entrez le caractere a traiter : ");
   scanf("%c",&c);
   if('A'<=c && c<='Z')
      c=c+('a'-'A');
   printf("Caractere apres le traitement : %c\n",c);

   return EXIT_SUCCESS;
}
```

Exercice 4

```
int main(void)
{  int a,b;
   int q,r;
   printf("Entrez le premier operande : ");
   scanf("%d",&a);
   printf("Entrez le second operande : ");
   scanf("%d",&b);
   q = a / b;
   r = a % b;
   printf("%d = %d*%d + %d\n",a,b,q,r);

   return EXIT_SUCCESS;
}
```

Exercice 5

```
int main()
{  int a,b;
   float resultat;
   printf("Entrez le premier operande : ");
   scanf("%d",&a);
   printf("Entrez le second operande : ");
   scanf("%d",&b);
   resultat=(float)a/b;
   printf("%d/%d=%f\n",a,b,resultat);

   return EXIT_SUCCESS;
}
```

Exercice 6

```
int main()
{  float nombre;
   printf("Entrez le nombre reel : ");
   scanf("%f",&nombre);
   nombre = (float)((int)(nombre*100))/100;
   printf("%f\n",nombre);

   return EXIT_SUCCESS;
}
```

Exercice 7

Toute expression possède une valeur. La valeur d'une expression d'affectation est égale à la valeur affectée. Par exemple, la valeur de l'expression `x=7` est : 7. Son type est celui de la variable recevant la valeur, i.e. `int` dans cet exemple.

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 04 | instructions de contrôle

2 Test simple avec alternative

Exercice 1

```
int main(int argc, char **argv)
{  int a,b,c, max;
   printf("Entrez les valeurs de a, b et c (sous la forme a:b:c) : \n");
   scanf("%d:%d:%d",&a,&b,&c);
   max = a;
   if(max<b)
      max = b;
   if(max<c)
      max = c;
   printf("le plus grand des trois entiers est %d.\n",max);

   return EXIT_SUCCESS;
}
```

Exercice 2

```
int main(int argc, char **argv)
{     float a,b,c, delta, x1,x2;
   printf("Entrez les valeurs des coefficients a, b et c du trinome
                                                    (a doit etre non-nul)\n");
   printf("a=");scanf("%f",&a);
   printf("b=");scanf("%f",&b);
   printf("c=");scanf("%f",&c);
   delta = b*b-4*a*c;
   printf("L'equation %.2fx^2 + %.2fx + %.2f = 0 ",a,b,c);
   if(delta>0)
   {  printf("admet deux solutions : ");
      x1 = (-b-sqrt(delta))/(2*a);
      x2 = (-b+sqrt(delta))/(2*a);
      printf("x1 = %f et x2 = %f\n",x1,x2);
   }
   else if(delta==0)
   {  printf("admet exactement une solution : ");
      x1 = -b/a;
      printf("x = %f\n",x1);
   }
   else //if (delta<0)
   {  printf("n'admet pas de solution.\n");
   }

   return EXIT_SUCCESS;
}
```

3 Boucles simples

Exercice 3

```
1  int main(int argc, char **argv)
2  {   int x,y;

3      printf("Entrez l'entier a renverser : ");
4      scanf("%d",&x);
```

```
5       y=0;
6       while(x>0)
7       {   y = y*10 + x%10;
8           x = x/10;
9       }
10      printf("Resultat : %d\n",y);

11      return EXIT_SUCCESS;
12  }
```

Exercice 4

Tableau de situation de l'exercice précédent :

ligne	x	y
1	_	_
2	_	_
3	_	_
4	**1234**	_
5	1234	**0**
6	1234	0
7	1234	**4**
8	**123**	4

ligne	x	y
9	123	4
6	123	4
7	123	**43**
8	**12**	43
9	12	43
6	12	43
7	12	**432**
8	**1**	432

ligne	x	y
9	1	432
6	1	432
7	1	**4321**
8	**0**	4321
9	0	4321
6	0	4321
10	0	4321
11	0	4321

4 Boucles imbriquées

Exercice 5

```
#define N 10

int main(int argc, char **argv)
{   int i,p,somme=1;

    for (p=2;p<=N;p++)
    {   for(i=1;i<p;i++)
            printf("%3d +",i);
        somme=somme+p;
        printf("%3d = %3d \n",p,somme);
    }

    return EXIT_SUCCESS;
}
```

Exercice 6

```
int main(int argc, char **argv)
{   int i, n, p, somme;
    do
    {   printf("Entrez n ou une valeur<=0 pour terminer : ");
        scanf("%d",&n);
        somme=1;
        for (p=2;p<=n;p++)
        {   for(i=1;i<p;i++)
            printf("%3d +",i);
            somme=somme+p;
            printf("%3d = %3d \n",p,somme);
        }
    }
    while(n>0);
    printf("Le programme se termine.\n");

    return EXIT_SUCCESS;
}
```

5 Suite de Fibonacci

Exercice 7

```
int main(int argc, char **argv)
{  int n,i;
   long fib_i=1, fib_precedent=0,temp;
   printf("Entrez la valeur de n : ");
   scanf("%d",&n);
   printf("u%d=%d\n",0,0);
   printf("u%d=%d\n",1,1);
   for(i=2;i<=n;i++)
   {  temp = fib_i + fib_precedent;
      fib_precedent = fib_i;
      fib_i = temp;
      printf("u%d=%d\n",i,fib_i);
   }

   return EXIT_SUCCESS;
}
```

Exercice 8

```
int main(int argc, char **argv)
{  int n,i;
   double fib_i=1, fib_precedent=0,temp,rapport;
   printf("Entrez la valeur de n : ");
   scanf("%d",&n);
   printf("u%d=%d\n",0,0);
   printf("u%d=%d\n",1,1);
   for(i=2;i<=n;i++)
   {  temp = fib_i + fib_precedent;
      fib_precedent = fib_i;
      fib_i = temp;
      rapport = fib_i / fib_precedent;
      printf("u%d=%.1f\n",i,fib_i);
      printf("  u%d/u%d=%.15lf\n",i,i-1,rapport);
   }

   return EXIT_SUCCESS;
}
```

Exercice 9

```
#define PHI 1.61803398874

int main(int argc, char **argv)
{  int i=2;
   double fib_i=1, fib_precedent=0,temp,rapport,erreur;
   printf("u%d=%d\n",0,0);
   printf("u%d=%d\n",1,1);
   do
   {  temp = fib_i + fib_precedent;
      fib_precedent = fib_i;
      fib_i = temp;
      rapport = (fib_i/fib_precedent);
      erreur = rapport - PHI;
      printf("u%d=%.1f\n",i,fib_i);
      printf("u%d/u%d=%.15lf\n",i,i-1,rapport);
      printf("erreur : %.15lf\n",erreur);
      i++;
   }
   while(fabs(erreur)>1e-10);

   return EXIT_SUCCESS;
}
```

6 Devinette

Exercice 10

```
int main(int argc, char **argv)
{  srand(time(NULL));
   int n=1+rand()%100,m,score;
   printf("J'ai tire un nombre au hasard : essayez de le deviner !\n");
   printf("  Premiere tentative ? ");
   scanf("%d",&m);
   score = 1;
   while(n!=m)
   {  printf("  La valeur %d est trop ",m);
      if(m<n)
         printf("petite\n");
      else
         printf("grande\n");
      printf("  Tentative %d ? ",score);
      scanf("%d",&m);
      score++;
   }
   printf("  Oui, c'est bien ca (%d) ! Bravo, vous avez gagne !\n",n);
   printf("Votre score est : %d\n", score);

   return EXIT_SUCCESS;
}
```

La meilleure stratégie est d'adopter une approche dichotomique :

1. Considérer l'intervalle des valeurs possibles $[1; 100]$;
2. Proposer la valeur médiane de cet intervalle : 50 ;
3. Si le programme répond « trop grand », alors on recommence avec l'intervalle $[1; 49]$, s'il répond « trop petit », alors on recommence avec $[51; 100]$.
4. On s'arrête quand on atteint la bonne valeur.

À chaque itération, on divise le nombre de valeurs restantes par 2. Donc, dans le pire des cas, on va devoir effectuer $\lceil \log_2 100 \rceil$ itérations (c'est-à-dire : 7) avant de trouver la solution.

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 05 | crible d'Ératosthène

2 Approche naïve

Exercice 1

```
int main()
{  int n,diviseur=2;
   printf("Entrez le nombre naturel a tester : ");
   scanf ("%d",&n);fflush(stdin);
   while((n%diviseur)!=0 && diviseur<n)
      diviseur++;
   if(diviseur!=n)
      printf("%d n'est pas un nombre premier : il est divisible par
                                                          %d.\n",n,diviseur);
   else
      printf("%d est un nombre premier.\n",n);

   return EXIT_SUCCESS;
}
```

Exercice 2

```
int main()
{  int n,v,diviseur;
   printf("Entrez un entier naturel (>2) : ");
   scanf ("%d",&v);
   printf("%5d est un nombre premier.\n",1);
   printf("%5d est un nombre premier.\n",2);
   for(n=3;n<=v;n++)
   {  diviseur=2;
      while((n%diviseur)!=0 && diviseur<n)
         diviseur++;
      if(diviseur==n)
         printf("%5d est un nombre premier.\n",n);
   }

   return EXIT_SUCCESS;
}
```

Exercice 3

```
#define MAX 100 // nombre maximum de nombres premiers à calculer

int main()
{  int n,v,diviseur,i,premier[MAX];
   printf("Entrez un entier naturel (>2) : ");
   scanf ("%d",&v);
   premier[0] = 1;
   premier[1] = 2;
   i=2;  // emplacement du prochain nombre premier dans le tableau
   for(n=3;n<=v;n++)
   {  diviseur = 2;
      while((n%diviseur)!=0 && diviseur<n)
         diviseur++;
      if(diviseur==n)
      {  premier[i] = n; // mise a jour du tableau
         i++;  // on passe a la cellule suivante du tableau
      }
```

```
    }
    for(n=0;n<i;n++)
       printf("%5d est un nombre premier.\n",premier[n]);

    return EXIT_SUCCESS;
}
```

3 Crible d'Ératosthène

Exercice 4

Contrôlez vos résultats avec le programme précédent (algorithme naïf).

Exercice 5

```
#define MAX 100

int main()
{  int n,v,d,i,premier[MAX];
   printf("Entrez un entier naturel (>2) : ");
   scanf ("%d",&v);
   premier[0] = 1;
   premier[1] = 2;
   i = 2;
   for(n=3;n<=v;n++)
   {  d = 1;
      // on compare maintenant au contenu du tableau
      while(d<i && (n%premier[d])!=0 && premier[d]<=n)
         d++;
      // on verifie si on a depasse le dernier nombre
      // mis dans le tableau premier
      if(d==i)
      {  premier[i] = n;
         i++;
      }
   }
   for(n=0;n<i;n++)
      printf("%5d est un nombre premier.\n",premier[n]);

   return EXIT_SUCCESS;
 }
```

4 Amélioration

Exercice 6

Preuve de P_1 :

- $\Rightarrow$: L'implication est vraie par définition des nombres premiers
- $\Leftarrow$: Montrons la contraposée, i.e. : $n \notin P \Rightarrow \left(\exists p \in P \cap \left[2, \frac{n}{2}\right] : p \; divise \; n\right)$
  - n n'est pas premier, il est donc décomposable en un produit.
  - Soit p son plus petit facteur premier.
  - Il existe alors un entier $d \geq 2$ tel que $n = dp$, et on a donc $p = n/d$.
  - Comme $d \geq 2$, il vient : $p \leq n/2$, d'où $p \in \left[2; \frac{n}{2}\right]$.
  - Comme p est premier, on a finalement : $p \in P \cap \left[2; \frac{n}{2}\right]$.

Exercice 7

```
#define MAX 100

int main()
{  int n,v,d,limite,i,premier[MAX/2];
   printf("Entrez un entier naturel (>2) : ");
   scanf ("%d",&v);
```

```
      premier[0] = 1;
      premier[1] = 2;
      i = 2;
      for(n=3;n<=v;n++)
      {  d = 1;
         limite = n/2;
         while(d<i && (n%premier[d])!=0 && premier[d]<=limite)
            d++;
         if(premier[d]>limite || d==i)
         {  premier[i] = n;
            i++;
         }
      }
      for(n=0;n<i;n++)
         printf("%5d est un nombre premier.\n",premier[n]);

      return EXIT_SUCCESS;
   }
```

Remarque : ce programme peut encore être amélioré :

- Il est inutile de tester les entiers pairs.
- On peut aussi utiliser la propriété suivante :

$$P_2 : n \in P \Leftrightarrow \forall p \in P \cap \left[2; \sqrt{n}\right], p \; ne \; divise \; pas \; n$$

Cette propriété permet de diminuer le nombre de divisions effectuées par l'algorithme. Mais il faut prendre en compte le problème de transtypage puisque la valeur retournée par la fonction `sqrt()` définie dans la bibliothèque `math.h` est de type `float`.

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 06 | chaînes de caractères

2 Exercices

Exercice 1

```
int main()
{  char chaine[5];
   int i=0;
   printf("Entrez une chaine de caracteres : ");
   scanf("%s",chaine);
   while(chaine[i]!='\0')
      i++;
   printf("Il y a %d caracteres dans la chaine \"%s\"\n",i,chaine);

   return EXIT_SUCCESS;
}
```

Exercice 2

```
int main()
{  char chaine1[5],chaine2[5];
   int i=0;
   printf("Entrez une chaine de caracteres : ");
   scanf("%s",chaine1);
   while(chaine1[i]!='\0')
   {  chaine2[i] = chaine1[i];
      i++;
   }
   chaine2[i] = '\0';
   printf("La chaine 2 est : %s\n",chaine2);

   return EXIT_SUCCESS;
}
```

Si on utilise un tableau `chaine2` plus petit que `chaine1`, il n'y aura pas d'erreur de compilation. Par contre, il pourra y avoir des erreurs d'exécution, car on risque d'écrire dans une partie de la mémoire qui n'appartient pas au tableau. Si au contraire `chaine2` est plus grand que `chaine1`, alors ce problème n'apparait pas (à condition que la chaîne de caractères contienne bien un `'\0'` final et qu'elle ne dépasse pas elle-même de `chaine1`).

Exercice 3

```
int main()
{  char chaine1[5];
   char chaine2[5];
   char chaine3[9];
   int i,j;
   // saisie des chaines
   printf("Entrez la chaine 1 : ");
   scanf("%s",chaine1);
   printf("Entrez la chaine 2 : ");
   scanf("%s",chaine2);
   // concaténation des chaines
   i=0;j=0;
   while(chaine1[i]!='\0')
   {  chaine3[j]=chaine1[i];
      i++;
```

```
        j++;
    }
    i=0;
    while(chaine2[i]!='\0')
    {  chaine3[j]=chaine2[i];
       i++;
       j++;
    }
    chaine3[j]='\0';
    // affichage du résultat
    printf("La chaine 3 est leur concatenation : %s\n",chaine3);

    return EXIT_SUCCESS;
}
```

Le tableau `chaine3` doit au moins être aussi grand que la somme des tailles de `chaine1` et `chaine2`, car il doit contenir leurs caractères… moins 1 caractère, car il ne doit contenir qu'une seule fois `'\0'` (qui apparait deux fois : une dans `chaine1` et une autre dans `chaine2`).

Exercice 4

```
int main()
{  char chaine1[5], chaine2[5];
   int i=0;
   // saisie des chaines
   printf("Entrez la chaine 1 : ");
   scanf("%s",chaine1);
   printf("Entrez la chaine 2 : ");
   scanf("%s",chaine2);
   // comparaison des chaines
   while(chaine1[i]==chaine2[i] && chaine1[i]!='\0')
      i++;
   // affichage du résultat
   printf("Resultat : ");
   if(chaine1[i]>chaine2[i])
      printf("%s>%s\n",chaine1,chaine2);
   else
      if(chaine1[i]<chaine2[i])
         printf("%s<%s\n",chaine1,chaine2);
      else
         printf("%s=%s\n",chaine1,chaine2);

   return EXIT_SUCCESS;
}
```

Exercice 5

```
int main()
{  char chaine1[] = "abcde";
   char chaine2[6];
   int longueur=0,i=0;
   // on calcule d'abord la longueur de la chaine
   while(chaine1[longueur]!='\0')
      longueur++;
   // puis on recopie à l'envers
   while(chaine1[i]!='\0')
   {  chaine2[longueur-1-i] = chaine1[i];
      i++;
   }
   // on rajoute le caractere de fin
   chaine2[longueur] = '\0';
   // on affiche le resultat
   printf("La chaine 1 est %s\n",chaine1);
   printf("La chaine 2 est son inversion : %s\n",chaine2);

   return EXIT_SUCCESS;
}
```

Exercice 6

```
int main()
{  char chaine1[] = "abcde",temp;
   int longueur=0,i=0;
   printf("La chaine 1 est %s\n",chaine1);
   // on calcule d'abord la longueur de la chaine
   while(chaine1[longueur]!='\0')
      longueur++;
   // puis on intervertit les caracteres un par un
   while(i<longueur/2)
   {  temp = chaine1[i];
      chaine1[i] = chaine1[longueur-1-i];
      chaine1[longueur-1-i] = temp;
      i++;
   }
   // on affiche le resultat
   printf("La chaine renversee est : %s\n",chaine1);

   return EXIT_SUCCESS;
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

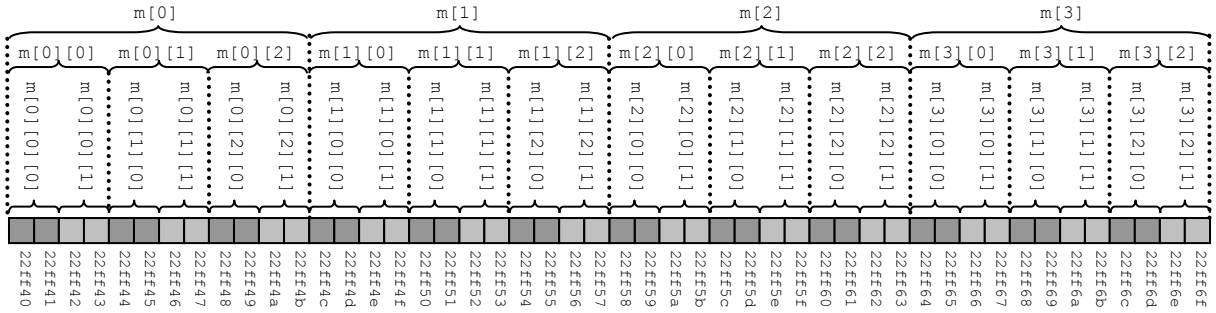
Damien Berthet & Vincent Labatut

Correction TP 07 | tableaux multidimensionnels

1 Occupation mémoire

Exercice 1

Occupation mémoire du tableau `short m[N][P][Q]` pour $N = 4$, $P = 3$ et $Q = 2$ (l'adresse de départ est choisie de façon arbitraire, ça pourrait être n'importe quelle valeur) :



Exercice 2

- Notons a l'adresse du tableau `m` et s la taille d'un `short` en mémoire. On a alors les formules suivantes :
  - adresse de `m[i]` : $a + (i \times P \times Q \times s)$.
  - adresse de `m[i][j]` : $a + (i \times P \times Q \times s) + (j \times Q \times s)$.
  - adresse de `m[i][j][k]` : $a + (i \times P \times Q \times s) + (j \times Q \times s) + (k \times s)$.
- Programme de vérification :

```
#define N 4
#define P 3
#define Q 2

int main()
{  short m[N][P][Q];
   int i,j,k,adresse;
   printf("Entrez i,j,k (avec les virgules) : ");
   scanf("%d,%d,%d",&i,&j,&k);
   adresse = (int) m;

   printf("Adresse du tableau m : %8x\n",adresse);

   printf("Adresse reelle de   m[%d] : %p\n",i,&m[i]);
   adresse = adresse + i*P*Q*sizeof(short);
   printf("Adresse calculee de m[%d] : %8x\n",i,adresse);

   printf("Adresse reelle de   m[%d][%d] : %p\n",i,j,&m[i][j]);
   adresse = adresse + j*Q*sizeof(short);
   printf("Adresse calculee de m[%d][%d] : %8x\n",i,j,adresse);

   printf("Adresse reelle de   m[%d][%d][%d] : %p\n",i,j,k,&m[i][j][k]);
   adresse = adresse + k*sizeof(short);
   printf("Adresse calculee de m[%d][%d][%d] : %8x\n",i,j,k,adresse);

    return EXIT_SUCCESS;
```

```
}
```

- **Remarque :** Le type de `m` n'est pas `int`, mais `short***`. Pour cette raison il est nécessaire de transtyper (cast) sa valeur pour l'affecter à la variable `adresse` (qui elle est de type `int`). Pour être vraiment correct, il faudrait utiliser le type `void*` plutôt que `int`, mais cela implique des notions pas encore vues en cours.

Exercice 3

```
#define N 4
#define P 3
#define Q 2

int main()
{  short m[N][P][Q];
   int i,j,k,compteur=0,t;
   for(i=0;i<N;i++)
   {  for(j=0;j<P;j++)
      {  for(k=0;k<Q;k++)
         {  m[i][j][k] = compteur;
            compteur = compteur + sizeof(short);
         }
      }
   }
   printf("Entrez i,j,k pour vérification (avec les virgules) : ");
   scanf("%d,%d,%d",&i,&j,&k);
   t = i*P*Q*sizeof(short) + j*Q*sizeof(short) + k*sizeof(short);
   printf("m[%d][%d][%d]:\n",i,j,k);
   printf("\tValeur theorique : %d\n",t);
   printf("\tValeur effective : %d\n",m[i][j][k]);

    return EXIT_SUCCESS;
}
```

2 Opérations matricielles

Exercice 4

```
int main()
{  int m[N][N] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
   int i,j;
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         printf(" %2d",m[i][j]);
      printf("\n");
   }

   return EXIT_SUCCESS;
}
```

Exercice 5

```
int main()
{  int m[N][N] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
   int i,j;
   int res[N][N],s=2;

   printf("Matrice originale :\n");
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         printf(" %2d",m[i][j]);
      printf("\n");
   }

   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         res[i][j] = m[i][j]*s;
   }
```

```
      printf("Résultat de la multiplication par %d :\n",s);
      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            printf(" %2d",res[i][j]);
         printf("\n");
      }

       return EXIT_SUCCESS;
   }
```

Exercice 6

```
   int main()
   {  int m[N][N] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
      int i,j;
      int res[N][N];

      printf("Matrice originale :\n");
      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            printf(" %2d",m[i][j]);
         printf("\n");
      }

      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            res[j][i] = m[i][j];
      }
      printf("Résultat de la transposition :\n");
      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            printf(" %2d",res[i][j]);
         printf("\n");
      }

       return EXIT_SUCCESS;
   }
```

Exercice 7

```
   int main()
   {  int m1[N][N] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
      int m2[N][N] = {{1,1,1,1},{2,2,2,2},{3,3,3,3},{4,4,4,4}};
      int res[N][N];
      int i,j;

      printf("Matrice originale 1 :\n");
      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            printf(" %2d",m1[i][j]);
         printf("\n");
      }
      printf("Matrice originale 2 :\n");
      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            printf(" %2d",m2[i][j]);
         printf("\n");
      }

      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            res[i][j] = m1[i][j] + m2[i][j];
      }
      printf("Résultat de l'addition :\n");
      for(i=0;i<N;i++)
      {  for(j=0;j<N;j++)
            printf(" %2d",res[i][j]);
         printf("\n");
      }
```

```
    return EXIT_SUCCESS;
}
```

Exercice 8

```
int main()
{  int m1[N][N] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
   int m2[N][N] = {{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1}};
// int m2[N][N] = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
   int res[N][N];
   int i,j,k;

   printf("Matrice originale 1 :\n");
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         printf(" %2d",m1[i][j]);
      printf("\n");
   }
   printf("Matrice originale 2 :\n");
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         printf(" %2d",m2[i][j]);
      printf("\n");
   }

   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
      {  res[i][j] = 0;
         for(k=0;k<N;k++)
            res[i][j] = res[i][j] + m1[i][k]*m2[k][j];
      }
   }
   printf("Résultat de la multiplication :\n");
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         printf(" %2d",res[i][j]);
      printf("\n");
   }

    return EXIT_SUCCESS;
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 08 | diagrammes en mode texte

2 Carrés

Exercice 1

```
void affiche_ligne(int n, char car)
{  int i;
   for(i=0;i<n;i++)
      printf("%c ",car);
   printf("\n");
}
```

Exercice 2

```
void affiche_carre(int cote, char car)
{  int i;
   for(i=0;i<cote;i++)
      affiche_ligne(cote,car);
}
```

3 Triangles

Exercice 3

```
void affiche_triangle(int cote, char car)
{  int i;
   for(i=1;i<=cote;i++)
      affiche_ligne(i,car);
}
```

Exercice 4

```
void affiche_triangle2(int cote, char car, int direction)
{  int i;
   if(direction==0)
      affiche_triangle(cote,car);
   else
   {  for(i=cote;i>=1;i--)
         affiche_ligne(i,car);
   }
}
```

Exercice 5

```
void affiche_grand_triangle(int cote, char car)
{  affiche_triangle2(cote,car,0);
   affiche_triangle2(cote-1,car,1);
}
```

4 Croix

Exercice 6

```
void affiche_ligne2(int n1, int n2, char car)
```

```
{  int i;
   for(i=0;i<n1;i++)
      printf("  ");
   affiche_ligne(n2,car);
}
```

Exercice 7

```
void affiche_croix(int cote, char car)
{  int rep = cote-1;
   int i;
   for(i=0;i<rep;i++)
      affiche_ligne2(rep,1,car);
   affiche_ligne(rep*2+1,car);
   for(i=0;i<cote-1;i++)
      affiche_ligne2(rep,1,car);
}
```

Exercice 8

```
void affiche_croix2(int cote, int largeur, char car)
{  int rep = cote-1-largeur/2;
   int i;
   for(i=0;i<rep;i++)
      affiche_ligne2(rep,largeur,car);
   for(i=0;i<largeur;i++)
      affiche_ligne((cote-1)*2+1,car);
   for(i=0;i<rep;i++)
      affiche_ligne2(rep,largeur,car);
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 09 | introduction à la SDL

2 Fenêtre graphique

Exercice 2

- Modification du titre de la fenêtre dans `main.c` :

```
initialise_fenetre("Emre Emreoglu");
```

- Modification des constantes dans `graphisme.h` :

```
/** Largeur de la fenetre graphique */
#define FENETRE_LARGEUR 600
/** Hauteur de la fenetre graphique */
#define FENETRE_HAUTEUR 600
```

3 Représentation en mémoire

Exercice 3

```
#define C_BLEU      convertis_rvb(0,0,255)
#define C_ORANGE    convertis_rvb(237,127,16)
#define C_JAUNE     convertis_rvb(255,255,0)
#define C_BLANC     convertis_rvb(255,255,255)
#define C_ROUGE     convertis_rvb(255,0,0)
#define C_GRIS      convertis_rvb(125,125,125)
#define C_VERT      convertis_rvb(0,255,0)
#define C_NOIR      convertis_rvb(0,0,0)
```

4 Dessin dans la fenêtre

Exercice 4

```
void trace_carre(int x, int y, int c, Uint32 coul)
{  int i,j;

   for(i=x;i<x+c;i++)
   {  for(j=y;j<y+c;j++)
      {  allume_pixel(i, j, coul);
      }
   }
}
```

Exercice 5

```
   int cote = 100;
   int x = (FENETRE_LARGEUR-cote)/2;
   int y = (FENETRE_HAUTEUR-cote)/2;
   trace_carre(x, y, cote, C_BLANC);
   rafraichis_fenetre();
   attends_touche();
```

Exercice 6

```
void trace_figure()
{  int cote = 100;
   int marge = 10;

   trace_carre(FENETRE_LARGEUR-marge-cote, marge, cote, C_ROUGE);
```

```
    trace_carre((FENETRE_LARGEUR-cote)/2, (FENETRE_HAUTEUR-cote)/2, cote,
                                                                C_BLANC);
    trace_carre(FENETRE_LARGEUR-marge-cote, FENETRE_HAUTEUR-marge-cote, cote,
                                                                C_VERT);
    trace_carre(marge, marge, cote, C_BLEU);
    trace_carre(marge, FENETRE_HAUTEUR-marge-cote, cote, C_JAUNE);
    trace_carre(marge, (FENETRE_HAUTEUR-cote)/2, cote, C_ORANGE);
    trace_carre(FENETRE_LARGEUR-cote-marge, (FENETRE_HAUTEUR-cote)/2, cote,
                                                                C_GRIS);

    rafraichis_fenetre();
}
```

5 Animation de la fenêtre

Exercice 7

```
void anime_figure()
{  int cote = 100;
   int marge = 10;

   trace_carre(FENETRE_LARGEUR-marge-cote, marge, cote, C_ROUGE);
   rafraichis_fenetre();attends_delai(DELAI);efface_fenetre();

   trace_carre((FENETRE_LARGEUR-cote)/2, (FENETRE_HAUTEUR-cote)/2, cote,
                                                                C_BLANC);
   rafraichis_fenetre();attends_delai(DELAI);efface_fenetre();

   trace_carre(FENETRE_LARGEUR-marge-cote, FENETRE_HAUTEUR-marge-cote, cote,
                                                                C_VERT);
   rafraichis_fenetre();attends_delai(DELAI);efface_fenetre();

   trace_carre(marge, marge, cote, C_BLEU);
   rafraichis_fenetre();attends_delai(DELAI);efface_fenetre();

   trace_carre(marge, FENETRE_HAUTEUR-marge-cote, cote, C_JAUNE);
   rafraichis_fenetre();attends_delai(DELAI);efface_fenetre();

   trace_carre(marge, (FENETRE_HAUTEUR-cote)/2, cote, C_ORANGE);
   rafraichis_fenetre();attends_delai(DELAI);efface_fenetre();

   trace_carre(FENETRE_LARGEUR-cote-marge, (FENETRE_HAUTEUR-cote)/2, cote,
                                                                C_GRIS);
   rafraichis_fenetre();attends_delai(DELAI);efface_fenetre();

   trace_carre(FENETRE_LARGEUR-marge-cote, marge, cote, C_ROUGE);
   rafraichis_fenetre();
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 10 | modification des couleurs

1 Préparation

Exercice 1

```
SDL_Surface *photo, *gris, *modif;
initialise_fenetre("Modification des couleurs");
photo = charge_image("album/image1.bmp");
dessine_surface(0, 0, photo);
```

2 Niveaux de gris

Exercice 2

```
SDL_Surface* convertis_gris(SDL_Surface* source)
{  int x,y;
   Uint8 r,v,b,m;
   Uint32 coul;

   // on cree la nouvelle surface (vide)
   SDL_Surface* destination = initialise_surface(source->w, source->h);

   // on recopie chaque pixel, en niveaux de gris
   for(x=0;x<source->w;x++)
   {  for(y=0;y<source->h;y++)
      {  // on recupere les composantes du pixel
         composantes_pixel_surface(source, x, y,&r, &v, &b);

         // on calcule leur moyenne
         m = (Uint8)((r+v+b)/3);

         // on cree le nouveaux code couleur
         coul = convertis_rvb(m,m,m);
         // on recopie dans la nouvelle surface
         allume_pixel_surface(destination, x, y, coul);
      }
   }

   return destination;
}
```

Exercice 3

```
gris = convertis_gris(photo);
sauve_surface(gris,"album/image1.gris.bmp");
dessine_surface(photo->w, 0, gris);
attends_touche();
```

3 Balance des couleurs

Exercice 4

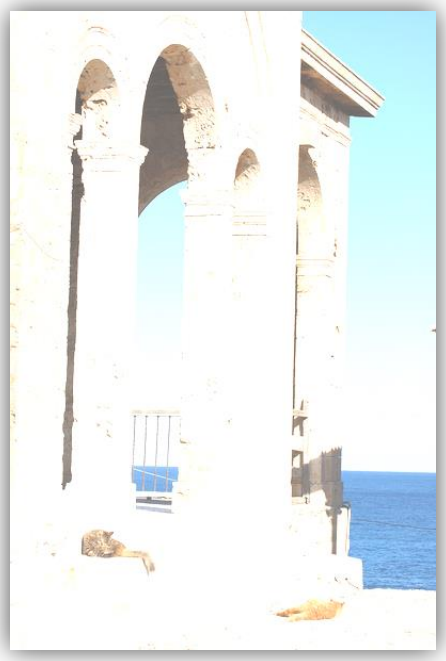
```
Uint8 seuil(Uint8 x, int y)
{  if(x+y>255)
      x = 255;
   else if(x+y<0)
```

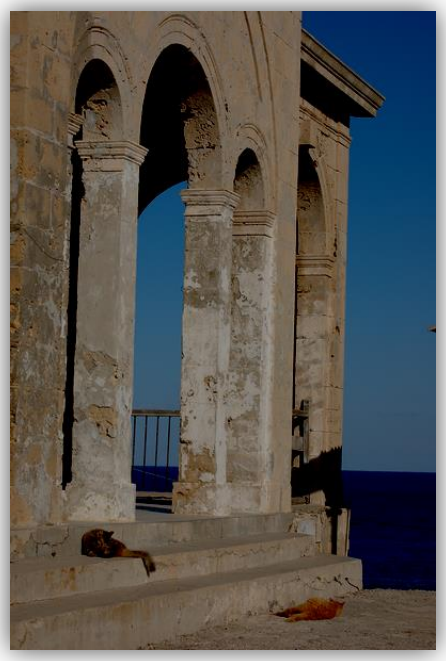
```
        x = 0;
    else
        x = x + y;
    return x;
}
```

Exercice 5

```
SDL_Surface* modifie_couleurs(SDL_Surface* source, int delta_r,
                                                    int delta_v, int delta_b)
{   int x,y;
    Uint8 r,v,b;
    Uint32 coul;

    // on cree la nouvelle surface (vide)
    SDL_Surface* destination = initialise_surface(source->w, source->h);

    // on recopie chaque pixel, avec sa couleur modifiee
    for(x=0;x<source->w;x++)
    {   for(y=0;y<source->h;y++)
        {   // on recupere les composantes du pixel
            composantes_pixel_surface(source, x, y,&r, &v, &b);

            // on applique les modifications
            r = seuil(r, delta_r);
            v = seuil(v, delta_v);
            b = seuil(b, delta_b);

            // on cree le nouveaux code couleur
            coul = convertis_rvb(r,v,b);
            // on recopie dans la nouvelle surface
            allume_pixel_surface(destination, x, y, coul);
        }
    }

    return destination;
}
```

Lorsqu'on applique la même modification aux trois composantes, on augmente ou diminue la luminosité, suivant que la modification est positive ou négative. Ainsi, pour les valeurs $(100,100,100)$ et $(-100,-100,-100)$ proposées dans le sujet, on obtient les images suivantes :





Exercice 6

```
    // modif = modifie_couleurs(photo, 100, 0, 0);
    // modif = modifie_couleurs(photo, 100, 100, 100);
       modif = modifie_couleurs(photo, -100, -100, -100);
       sauve_surface(modif,"album/image1.modif.bmp");
       dessine_surface(photo->w, 0, modif);
       attends_touche();
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 11 | passage de paramètres

1 Expérimentations

Exercice 1

Fonction de test :

```
void test1(int x)
{  printf("@test1 : la valeur de x est %d\n",x);
   printf("@test1 : l'adresse de x est %p\n",&x);
}
```

Fonction `main` :

```
int main()
{  int x = 12;
   printf("@main : l'adresse de x avant l'appel est %p\n",&x);
   printf("@main : la valeur de x avant l'appel est %d\n",x);
   test1(x);
   printf("@main : l'adresse de x apres l'appel est %p\n",&x);
   printf("@main : la valeur de x apres l'appel est %d\n",x);

   return EXIT_SUCCESS;
}
```

Comme expliqué en cours, la variable `x` déclarée dans la fonction `main` et celle déclarée dans la fonction `test1` sont deux variables qui portent le même nom, mais elles sont bien distinctes. Elles ont donc des adresses différentes.

De plus, on sait que lors de l'initialisation d'un paramètre *formel*, on utilise la valeur du paramètre *effectif* correspondant, donc la valeur du `x` de `test1` est une copie de la valeur du `x` de `main`.

Exercice 2

```
int main()
{  int n = 12;
   printf("@main : l'adresse de n avant l'appel est %p\n",&n);
   printf("@main : la valeur de n avant l'appel est %d\n",n);
   test1(n);
   printf("@main : l'adresse de n apres l'appel est %p\n",&n);
   printf("@main : la valeur de n apres l'appel est %d\n",n);

   return EXIT_SUCCESS;
}
```

On observe la même chose que pour l'exercice précédent, et ce pour les mêmes raisons. On peut remarquer qu'ici, le paramètre effectif et le paramètre formel correspondent à des variables d'identificateurs (noms) différents : ça ne change rien à la situation, puisque de toute façon ces deux variables sont distinctes, comme expliqué à l'exercice précédent.

Exercice 3

Fonction de test :

```
void test2(int x)
{  printf("@test2 : la valeur de x est %d\n",x);
   printf("@test2 : l'adresse de x est %p\n",&x);

   x = x / 2;
```

```
   printf("@test2 : la valeur de x apres la division est %d\n",x);
   printf("@test2 : l'adresse de x apres la division est %p\n",&x);
}
```

Fonction `main` :

```
int main()
{  int n = 12;
   printf("@main : l'adresse de n avant l'appel est %p\n",&n);
   printf("@main : la valeur de n avant l'appel est %d\n",n);
   test2(n);
   printf("@main : l'adresse de n apres l'appel est %p\n",&n);
   printf("@main : la valeur de n apres l'appel est %d\n",n);

   return EXIT_SUCCESS;
}
```

Exercice 4

Fonction de test :

```
int test3(int x)
{  int resultat;

   printf("@test3 : la valeur de x est %d\n",x);
   printf("@test3 : l'adresse de x est %p\n",&x);

   resultat = x / 2;

   printf("@test3 : le resultat de la division est %d\n",resultat);

   return resultat;
}
```

Fonction `main` :

```
int main()
{  int n = 12;
   printf("@main : l'adresse de n avant l'appel est %p\n",&n);
   printf("@main : la valeur de n avant l'appel est %d\n",n);
   n = test3(n);
   printf("@main : l'adresse de n apres l'appel est %p\n",&n);
   printf("@main : la valeur de n apres l'appel est %d\n",n);

   return EXIT_SUCCESS;
}
```

Exercice 5

Fonction de test :

```
void test4(int x, int* resultat)
{  printf("@test4 : la valeur de x est %d\n",x);
   printf("@test4 : l'adresse de x est %p\n",&x);
   printf("@test4 : l'adresse indiquee par le parametre resultat est %p\n",
                                                                  resultat);
   printf("@test4 : la valeur situee a cette adresse est %d\n",*resultat);

   *resultat = x / 2;

   printf("@test4 : la valeur de x apres la division est %d\n",x);
   printf("@test4 : l'adresse de x apres la division est %p\n",&x);
   printf("@test4 : l'adresse indiquee par le parametre resultat apres la
                                                  division est %p\n",resultat);
   printf("@test4 : la valeur situee a cette adresse apres la division est
                                                        %d\n",*resultat);
}
```

Fonction `main` :

```
int main()
{  int n = 12, r;
   printf("@main : l'adresse de n avant l'appel est %p\n",&n);
```

```
    printf("@main : la valeur de n avant l'appel est %d\n",n);
    printf("@main : l'adresse de r avant l'appel est %p\n",&r);
    printf("@main : la valeur de r avant l'appel est %d\n",r);
    test4(n,&r);
    printf("@main : l'adresse de n apres l'appel est %p\n",&n);
    printf("@main : la valeur de n apres l'appel est %d\n",n);
    printf("@main : l'adresse de r apres l'appel est %p\n",&r);
    printf("@main : la valeur de r apres l'appel est %d\n",r);

    return EXIT_SUCCESS;
}
```

Il est inutile d'initialiser la variable `r` dans la fonction `main`, car sa valeur sera de toute façon modifiée par la fonction `test4`, lors de l'affectation `*resultat=x/2`. Sa valeur avant ça n'a aucune importance dans ce programme.

L'adresse de `r` dans `main` et l'adresse indiquée par `resultat` dans `test4` sont les mêmes. C'est normal, puisque `resultat` est un paramètre passé par valeur. Autrement dit, il contient l'adresse d'une autre variable (ici `r`). Ceci est aussi visible lors de l'appel de la fonction test4 dans main, puisque son deuxième paramètre est `&r`, i.e. l'adresse de `r` (et non pas simplement `r`, i.e. la valeur de `r`).

Exercice 6

Fonction de test :

```
void test5(int* x)
{  printf("@test5 : l'adresse indiquee par le parametre x est %p\n",x);
   printf("@test5 : la valeur situee a cette adresse est %d\n",*x);

   *x = *x / 2;

   printf("@test5 : l'adresse indiquee par le parametre x apres la division est
                                                                    %p\n",x);
   printf("@test5 : la valeur situee a cette adresse apres la division est
                                                                    %d\n",*x);
}
```

Fonction `main` :

```
int main()
{  int n = 12;
   printf("@main : l'adresse de n avant l'appel est %p\n",&n);
   printf("@main : la valeur de n avant l'appel est %d\n",n);
   test5(&n);
   printf("@main : l'adresse de n apres l'appel est %p\n",&n);
   printf("@main : la valeur de n apres l'appel est %d\n",n);

   return EXIT_SUCCESS;
}
```

Ici, la variable x de la fonction test5 est à la fois utilisée comme entrée (elle contient l'adresses de la valeur que l'on veut diviser par 2) et comme sortie (elle contient l'adresse à laquelle on veut placer le résultat de la division par 2). À cause du premier de ces deux rôles (le rôle d'entrée de la fonction), la valeur située à l'adresse indiquée par `x` doit avoir été initialisée. Puisque c'est de l'adresse de `n` (variable de la fonction `main`) qu'il s'agit, alors cette variable doit avoir été obligatoirement initialisée. La correspondance entre `n` de `main` et `x` de `test5` est confirmée par le fait que l'adresse de `n` et celle indiquée par `x` sont les mêmes.

2 Fonctions mathématiques

Exercice 7

```
float vabs(float x)
{  float resultat = x;
   if(resultat<0)
      resultat = -x;
```

```
    return resultat;
}
```

Exercice 8

```
void distance(float x, float y, float* res)
{  float diff = x - y;
   *res = vabs(diff);
}
```

Exercice 9

Définition de la fonction :

```
void division_entiere(int x, int y, int *q, int *r)
{  *q = x / y;
   *r = x % y;
}
```

Appel depuis la fonction `main` :

```
int main()
{  int x=77,y=12,q,r;
   division_entiere(x, y, &q, &r);
   printf("%d/%d = %d modulo %d\n",x,y,q,r);

   return EXIT_SUCCESS;
}
```

Exercice 10

```
int calcule_racine(float a, float b, float c, float* r1, float* r2)
{  // on calcule d'abord le discriminant
   float delta = b*b - 4*a*c;
   float rdelta;
   // puis on determine le nombre de racines et on les calcule
   int code;
   if(delta<0)
      code = 0;
   else
   {  rdelta = sqrt(delta);
      if(delta==0)
      {  code = 1;
         *r1 = -b / (2*a);
      }
      else
      {  code = 2;
         *r1 = (-b - rdelta) / (2*a);
         *r2 = (-b + rdelta) / (2*a);
      }
   }
   // et enfin on renvoie le code
   return code;
}
```

Exercice 11

```
void affiche_racines(float a, float b, float c)
{  float r1, r2;
   int code = calcule_racine(a, b, c, &r1, &r2);

   printf("Traitement du polynome %.2fx^2 + %.2fx + %.2f = 0\n",a,b,c);
   if(code==0)
      printf("Il n'existe aucune racine relle pour ce polynome\n");
   else if(code==1)
      printf("Il n'existe qu'une seule racine reelle pour ce polynome :
                                                              r=%.2f\n",r1);
   else //code==2
      printf("Il existe deux racines reelles distinctes pour ce polynome:
                                              r1=%.2f et r2=%.2f\n",r1,r2);
}
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 12 | algorithme de Bresenham

3 Algorithme naïf

Exercice 1

- Fonction auxiliaire utilisée pour faciliter les calculs :

```
int arrondit(float x)
{  int resultat=(int)x;
   if(x-resultat>=0.5)
      resultat++;
   return resultat;
}
```

- Fonction dessinant le segment :

```
void trace_segment_naif(int xa, int ya, int xb, int yb)
{  int x = xa;
   int y = ya;
   int i;
   float a = ((float)ya-yb)/(xa-xb);
   float b = ya - a*xa;

   allume_pixel(x,y,C_BLANC);
   for(i=0;i<xb-xa;i++)
   {  x++;
      y = arrondit(a*x+b);
      allume_pixel(x,y,C_BLANC);
   }
}
```

4 Algorithme de Bresenham

Exercice 2

```
void trace_segment_bresenham1(int xa, int ya, int xb, int yb)
{  int x = xa;
   int y = ya;
   int i;
   int dx = xb-xa;
   int dy = yb-ya;
   int d = 2*dy - dx;

   allume_pixel(x,y,C_ROUGE);
   for(i=0;i<dx;i++)
   {  x++;
      if(d<0)
      {  y++;
         d = d + 2*dy;
      }
      else
         d = d + 2*(dy-dx);
      allume_pixel(x,y,C_ROUGE);
   }
}
```

5 Généralisation

Exercice 3

```
void trace_segment_bresenham2(int xa,int ya, int xb, int yb)
{  int i;
   int x = xa;
   int y = ya;
   int x_inc;   // increment de x
   int y_inc;   // increment de y
   int dx = xb - xa;
   int dy = yb - ya;
   int d;

   // initialisation
   if (dx>=0)
      x_inc = 1;
   else
      x_inc = -1;
   if(dy>0)
      y_inc = 1;
   else
      y_inc = -1;
   dx = x_inc*dx; // dx=abs(dx)
   dy = y_inc*dy; // dy=abs(dy)

   // tracé du segment
   allume_pixel(x,y,C_VERT);
   // 0<=abs(a)<=1 : suivant les colonnes
   if(dy<=dx)
   {  d = 2*dy - dx;
      for(i=0;i<dx;i++)
      {  if(d<=0)
            d = d + 2*dy;
         else
         {  d = d + 2*(dy-dx);
            y = y + y_inc;
         }
         x = x + x_inc;
         allume_pixel(x,y,C_VERT);
      }
   }
   // abs(a)>1 : suivant les lignes
   else
   {  d = 2*dx - dy;
      for(i=0;i<dy;i++)
      {  if(d<=0)
            d = d + 2*dx;
         else
         {  d = d + 2*(dx-dy);
            x = x + x_inc;
         }
         y = y + y_inc;
         allume_pixel(x,y,C_VERT);
      }
   }
}
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 13 | histogramme des couleurs

2 Génération de données de test

Exercice 1

```
#include "alea.h"

int main(int argc, char** argv)
{  int val;

   // exercice 1
   val = genere_entier(255);
   printf("Entier genere : %d\n",val);

   return 0;
}
```

Exercice 2

```
void genere_tableau(int tab[], int taille)
{  int i;
   for(i=0;i<taille;i++)
      tab[i] = genere_entier();
}
```

Exercice 3

- Dans `histogramme.c` :

```
void affiche_tableau(int tab[], int taille)
{  int i;
   for(i=0;i<taille;i++)
      printf("tab[%i]=%d\n", i, tab[i]);
}
```

- Dans la fonction `main` :

```
   // exercice 3
   genere_tableau(tab,taille);
   printf("Tableau genere :\n");
   affiche_tableau(tab,taille);
```

3 Définition de l'histogramme

Exercice 4

- Dans `histogramme.c` :

```
void decompte_valeurs(int tab[], int taille, int decompte[])
{  int i;

   // on initialise decompte
   for(i=0;i<VAL_MAX+1;i++)
      decompte[i] = 0;

   // on calcule les nombres d'occurrences
   for(i=0;i<taille;i++)
      decompte[tab[i]]++;
}
```

- Dans la fonction `main` :

```
    // exercice 4
    genere_tableau(tab,taille);
    printf("Tableau genere :\n");
    affiche_tableau(tab,taille);genere_tableau(tab,taille);
```

Exercice 5

- Dans `histogramme.c` :

```
void histogramme_horizontal(int decompte[], int taille_d)
{  int i, j;
   for(i=0;i<taille_d;i++)
   {  for(j=0;j<decompte[i];j++)
         printf("*");
      printf("\n");
   }
}
```

- Dans la fonction `main` :

```
    // exercice 5
    printf("Histogramme horizontal : \n");
    histogramme_horizontal(decompte,VAL_MAX+1);
    printf("\n");
```

Exercice 6

- Dans `histogramme.c` :

```
void histogramme_vertical(int decompte[], int taille_d)
{  int max=decompte[0],i,j;

   // on détermine la hauteur de l'histogramme
   for(i=0;i<taille_d;i++)
   {  if(decompte[i]>max)
         max = decompte[i];
   }

   // une ligne au-dessus
   for(j=0;j<2*taille_d+2;j++)
      printf("-");
   printf("\n");

   // on le trace
   for(i=max;i>0;i--)
   {  printf("|");
      for(j=0;j<taille_d;j++)
      {  if(decompte[j]>=i)
         {  printf("* ");
         }
         else
            printf("  ");
      }
      printf("|\n");
   }

   // une ligne au-dessous
   for(j=0;j<2*taille_d+2;j++)
      printf("-");
   printf("\n");
}
```

- Dans la fonction `main` :

```
    // exercice 6
    printf("Histogramme vertical : \n");
    histogramme_vertical(decompte,VAL_MAX+1);
    printf("\n");
```

4 Regroupement des valeurs

Exercice 7

- Dans `histogramme.c` :

```
void regroupe_classes(int decompte[], int nbr_classes, int classes[])
{  int c = (VAL_MAX+1)/nbr_classes;
   int i,j;

   // on initialise resultat
   for(i=0;i<nbr_classes;i++)
      classes[i] = 0;

   for(i=0;i<nbr_classes;i++)
   {  for(j=0;j<c;j++)
      {  classes[i] = classes[i] + decompte[i*c+j];
         //printf("classes[%d]=%d + decompte[%d]=%d\n",i,classes[i],i+j,
                                                          decompte[i*c+j]);
      }
   }
}
```

- Dans la fonction `main` :

```
   // exercice 7
   regroupe_classes(decompte, nbr_classes, classes);
   printf("Regroupement des classes :\n");
   affiche_tableau(classes,nbr_classes);
```

Exercice 8

```
   // exercice 8
   printf("Histogramme des classes : \n");
   histogramme_horizontal(classes,nbr_classes);
```

Exercice 9

```
void regroupe_etoiles(int classes[], int nbr_classes, int etoiles[],
                                                          int echelle)
{  int i;
   for(i=0;i<nbr_classes;i++)
      etoiles[i] = classes[i] / echelle;
}
```

5 Couleurs d'une image

Exercice 10

```
void extrais_composantes(SDL_Surface *surface, int tab_r[], int tab_v[],
                                                          int tab_b[])
{  int x,y;
   Uint8 r,v,b;

   for(x=0;x<surface->w;x++)
   {  for(y=0;y<surface->h;y++)
      {  // on recupere les composantes du pixel
         composantes_pixel_surface(surface,x,y,&r,&v,&b);
         // on les places dans les tableaux
         tab_r[x+y*surface->w] = (int)r;
         tab_v[x+y*surface->w] = (int)v;
         tab_b[x+y*surface->w] = (int)b;
      }
   }
}
```

Exercice 11

- Fonction `analyse_surface` :

```
void analyse_surface(SDL_Surface *surface)
```

```
{  int taille=surface->w*surface->h;
   int tab_r[taille],tab_v[taille],tab_b[taille];
   int dec_r[VAL_MAX+1],dec_v[VAL_MAX+1],dec_b[VAL_MAX+1];
   int nbr_classes=32, echelle=1000;
   int cl_r[nbr_classes],cl_v[nbr_classes],cl_b[nbr_classes];
   int et_r[nbr_classes],et_v[nbr_classes],et_b[nbr_classes];

   // extrait composantes
   extrais_composantes(surface, tab_r, tab_v, tab_b);

   // decompte valeurs
   decompte_valeurs(tab_r, taille, dec_r);
   decompte_valeurs(tab_v, taille, dec_v);
   decompte_valeurs(tab_b, taille, dec_b);

   // regroupe par classe
   regroupe_classes(dec_r, nbr_classes, cl_r);
   regroupe_classes(dec_v, nbr_classes, cl_v);
   regroupe_classes(dec_b, nbr_classes, cl_b);

   // met a l'echelle
   regroupe_etoiles(cl_r, nbr_classes, et_r, echelle);
   regroupe_etoiles(cl_v, nbr_classes, et_v, echelle);
   regroupe_etoiles(cl_b, nbr_classes, et_b, echelle);

   // affiche histogrammes
   printf("L'histogramme pour le rouge est :\n");
   histogramme_vertical(et_r,nbr_classes);
   printf("L'histogramme pour le vert est :\n");
   histogramme_vertical(et_v,nbr_classes);
   printf("L'histogramme pour le bleu est :\n");
   histogramme_vertical(et_b,nbr_classes);
}
```

- Dans la fonction `main` :

```
SDL_Surface *photo;

initialise_fenetre("Histogrammes des couleurs");
photo = charge_image("album\\image1.bmp");
analyse_surface(photo);
attends_touche();
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 14 | propriétés arithmétiques

1 Diviseurs d'un nombre

Exercice 1

```
void calcule_diviseurs(int n, int diviseurs[N], int *d)
{  int i;
   *d = 0;
   for(i=1;i<=n;i++)
   {  if(n%i == 0)
      {  diviseurs[*d] = i;
         (*d)++;
      }
   }
}
```

Exercice 2

```
int additionne_diviseurs_propres(int n)
{  int i,diviseurs[N],d,resultat=0;
   calcule_diviseurs(n,diviseurs,&d);
   for(i=0;i<d-1;i++)
      resultat = resultat + diviseurs[i];
   return resultat;
}
```

Exercice 3

```
int additionne_diviseurs(int n)
{  int resultat = additionne_diviseurs_propres(n) + n;
   return resultat;
}
```

Exercice 4

```
void affiche_sommes_diviseurs_propres(int max)
{  int i,s;

   // on affiche les valeurs n
   printf("n    :");
   for(i=2;i<=max;i++)
      printf(" %02d",i);
   printf("\n");

   // on affiche les valeurs s(n)
   printf("s(n) :");
   for(i=2;i<=max;i++)
   {  s = additionne_diviseurs_propres(i);
      printf(" %02d",s);
   }
   printf("\n");
}
```

2 Nombres parfaits, amicaux et sublimes

Exercice 5

```
int calcule_abondance(int n)
```

```
{  int resultat = additionne_diviseurs_propres(n) - n;
   return resultat;
}
```

Exercice 6

```
int est_parfait(int n)
{  int resultat = calcule_abondance(n) == 0;
   return resultat;
}
```

Exercice 7

```
int sont_amicaux(int n, int p)
{  int sn = additionne_diviseurs_propres(n);
   int sp = additionne_diviseurs_propres(p);
   int resultat = sn==p && sp==n;
   return resultat;
}
```

Exercice 8

```
int est_sublime(int n)
{  int diviseurs[N],d,resultat,sigma;
   calcule_diviseurs(n,diviseurs,&d);
   sigma = additionne_diviseurs(n);
   resultat = est_parfait(d) && est_parfait(sigma);
   return resultat;
}
```

3 Nombres abondants et déficients

Exercice 9

```
int est_abondant(int n)
{  int a = calcule_abondance(n);
   int resultat = a > 0;
   return resultat;
}

int est_deficient(int n)
{  int a = calcule_abondance(n);
   int resultat = a < 0;
   return resultat;
}
```

Exercice 10

```
void affiche_nombres(int max, int mode)
{  int i,j=1,test;
   for(i=2;i<max;i++)
   {  if(mode<0)
         test = est_deficient(i);
      else if(mode==0)
         test = est_parfait(i);
      else //if(mode>0)
         test = est_abondant(i);
      if(test)
      {  printf("%3d. %d\n",j,i);
         j++;
      }
   }
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 15 | algorithmes pour l'arithmétique

2 Division euclidienne

Exercice 1

On applique l'algorithme du sujet aux entiers $a = 57$ et $b = 11$:

- **Initialisation** : $q = 0$ et $r = 57$
- **Itérations** : tant que $r \geq b$
  - $q = 1$ et $r = 57 - 11 = 46$
  - $q = 2$ et $r = 46 - 11 = 35$
  - $q = 3$ et $r = 35 - 11 = 24$
  - $q = 4$ et $r = 24 - 11 = 13$
  - $q = 5$ et $r = 13 - 11 = 2$
- **Terminaison** :
  - On a bien : $57 = 5 \times 11 + 2$ et $0 \leq 2 < 11$

Exercice 2

```
void division_euclidienne(int a, int b, int* q, int* r)
{   *q=0;
    *r=a;

    while(*r>=b)
    {   *q=*q+1;
        *r=*r-b;
    }
}
```

Exercice 3

Un invariant de boucle est une propriété qui reste vraie à chaque itération de la boucle considérée. Ici, la propriété $a = bq + r$ est un invariant de la boucle contenue dans l'algorithme étudié.

Notons q_k et r_k les valeurs de q et r après k itérations, vérifions cette propriété par récurrence sur le nombre k d'itérations :

- Pour $k = 0$, on a :
  - $b \times q_0 + r_0 = b \times 0 + a = a$
  - La propriété est donc bien vérifiée pour $k = 0$
- Supposons la propriété vraie après k itérations,
  - On a alors : $a = bq_k + r_k$
  - Calculons $bq_{k+1} + r_{k+1} = b(q_k + 1) + (r_k - b) = \mathrm{b}q_k + r_k$
  - Donc, d'après l'hypothèse de récurrence, $bq_{k+1} + r_{k+1} = a$, et la propriété est vraie pour $k + 1$

En conclusion, la propriété d'invariant de boucle est donc bien vérifiée quel que soit le nombre k d'itérations effectuées.

3 Plus grand commun diviseur

Exercice 4

Appliquons l'algorithme du sujet aux entiers $a = 57$ et $b = 11$

- **Initialisation** :
  - $(d, u, v) = (57,1,0)$
  - $(d', u', v') = (11,0,1)$
- **Itérations :**
  - Itération 1 :
    - $57 = 11 \times 5 + 2$
    - $(d, u, v) = (11,0,1)$
    - $(d', u', v') = (2,1 - 5 \times 0,0 - 5 \times 1) = (2,1, -5)$
  - Itération 2 :
    - $11 = 2 \times 5 + 1$
    - $(d, u, v) = (2,1, -5)$
    - $(d', u', v') = (1,0 - 5 \times 1,1 - 5 \times (-5)) = (1, -5,26)$
  - Itération 3 :
    - $2 = 1 \times 2 + 0$
    - $(d, u, v) = (1, -5,26)$
    - $(d', u', v') = (0,1 - 2 \times (-5), -5 - 2 \times 26) = (0,10, -57)$
- **Terminaison :**
  - $1 = 57 \times (-5) + 11 \times 26 = \text{pgcd}(57,11)$

Exercice 5

```
void pgcd(int a, int b, int* u, int* v, int* d)
{  int dp=b, up=0, vp=1;
   int temp;
   int r, q;

   // initialisation
   *d = a;
   *u = 1;
   *v = 0;

   // iteration
   while(dp !=0)
   {  division_euclidienne(*d, dp, &q, &r);
      *d = dp;
      dp = r;
      temp = up;
      up = *u - q*temp;
      *u = temp;
      temp = vp;
      vp = *v - q*temp;
      *v = temp;
   }
}
```

Exercice 6

Pour tout k, d'_{k+1} est le reste de la division de d_k par d'_k et donc $d'_{k+1} < d'_k$. La suite des valeurs de d'_k est donc une suite d'entiers positifs strictement décroissante. La valeur de d'_k est donc nulle après un nombre fini k d'itérations et l'algorithme se termine.

Exercice 7

Vérifions que la propriété suivante est un invariant de boucle de l'algorithme :

$$\begin{cases} au + bv = d \\ au' + bv' = d' \\ \text{pgcd}(d, d') = \text{pgcd}(a, b) \end{cases}$$

On note (d'_k, u'_k, v'_k) et (d_k, u_k, v_k) les valeurs respectives de (d', u', v') et (d, u, v) après k itérations.

- Pour $k = 0$:
  - La propriété est vérifiée puisque $(d_0, u_0, v_0) = (a, 1, 0)$ et $(d'_0, u'_0, v'_0) = (b, 0, 1)$, donc :

$$\begin{cases} au_0 + bv_0 = d_0 = a \\ au'_0 + bv'_0 = d'_0 = b \\ \text{pgcd}(d_0, d'_0) = \text{pgcd}(a, b) \end{cases}$$

- On suppose la propriété vraie au après k itérations, c'est à dire :

$$\begin{cases} au_k + bv_k = d_k \\ au'_k + bv'_k = d'_k \\ \text{pgcd}(d_k, d'_k) = \text{pgcd}(a, b) \end{cases}$$

  - On note $d_k = qd'_k + r$ le résultat de la division de d_k par d'_k, et on a :

$$\begin{cases} (d_{k+1}, u_{k+1}, v_{k+1}) = (d'_k, u'_k, v'_k) \\ (d'_{k+1}, u'_{k+1}, v'_{k+1}) = (r, u_k - qu'_k, v_k - qv'_k) \end{cases}$$

  - Donc :

$$\begin{cases} au_{k+1} + bv_{k+1} = au'_k + bv'_k = d'_k = d_{k+1} \\ au'_{k+1} + bv'_{k+1} = a(u_k - qu'_k) + b(v_k - qv'_k) = d_k - qd'_k = r = d'_{k+1} \end{cases}$$

  - De plus :

$$\text{pgcd}(d_{k+1}, d'_{k+1}) = \text{pgcd}(d'_k, r) = \text{pgcd}(d'_k, d_k) = \text{pgcd}(a, b)$$

On a donc démontré que la propriété d'invariant de boucle est donc vraie pour tout nombre k d'itérations.

Exercice 8

Si $d'_k = 0$, alors $\text{pgcd}(d_k, d'_k) = \text{pgcd}(d_k, 0) = d_k = \text{pgcd}(a, b)$ d'après la propriété d'invariant de boucle. La valeur obtenue pour le pgcd à la fin de cet algorithme est donc correcte.

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 16 | bibliothèque `chaines`

2 Fonctions existantes

Exercice 4

```
int mesure_chaine(char* chaine)
{  int resultat = 0;
   while(chaine[resultat]!='\0')
      resultat++;
   return resultat;
}
```

Exercice 5

```
void copie_chaine(char* chaine1, char* chaine2)
{  int i = 0;
   while(chaine1[i]!='\0')
   {  chaine2[i] = chaine1[i];
      i++;
   }
   chaine2[i] = '\0';
}
```

Exercice 6

```
int compare_chaines(char* chaine1, char* chaine2)
{  int resultat, i=0;
   // comparaison des chaines
   while(chaine1[i]==chaine2[i] && chaine1[i]!='\0')
      i++;

   // calcul du résultat
   if(chaine1[i]>chaine2[i])
      resultat = +1;
   else
      if(chaine1[i]<chaine2[i])
         resultat = -1;
      else
         resultat = 0;
   return resultat;
}
```

Exercice 7

```
void inverse_chaine(char* chaine)
{  int i = 0;
   char temp;

   // on calcule d'abord la longueur de la chaine
   int longueur = mesure_chaine(chaine);

   // puis on intervertit les caracteres un par un
   while(i<longueur/2)
   {  temp = chaine[i];
      chaine[i] = chaine[longueur-1-i];
      chaine[longueur-1-i] = temp;
      i++;
   }
```

```
}
```

3 Nouvelles fonctions

Exercice 8

```
void supprime_majuscules(char* chaine)
{  int i,j=0;
   for(i = 0; chaine[i] != '\0'; i++)
   {  if(!('A'<= chaine[i] && chaine[i]<='Z'))
      {  chaine[j] = chaine[i];
         j++;
      }
   }
   chaine[j] = '\0';
}
```

Exercice 9

```
void remplace_majuscules(char* chaine)
{  int i;
   for(i = 0; chaine[i] != '\0'; i++)
   {  if('A'<= chaine[i] && chaine[i]<='Z')
         chaine[i] = chaine[i] - 'A' + 'a';
   }
}
```

Exercice 10

```
int compte_espaces(char *chaine)
{  int i = 0;
   int resultat = 0;

   while(chaine[i]!= '\0')
   {  if(chaine[i]==' ')
         resultat++;
      i++;
   }
   return resultat;
}
```

Exercice 11

```
int compte_mots(char *chaine)
{  int i = 0;
   int resultat = 0;

   // espaces en debut de chaine
   while(chaine[i]==' ')
      i++;

   // espaces dans la chaine
   while(chaine[i]!='\0')
   {  if(chaine[i]==' ' && chaine[i-1]!=' ')
         resultat++;
      i++;
   }

   // dernier caractere
   if(chaine[i-1]!=' ')
      resultat++;

   return resultat;
}
```

Exercice 12

```
int est_prefixe(char* chaine1, char* chaine2)
{  int i1=0,i2=0;
```

```
    int resultat;

    while(chaine1[i1]!='\0' && chaine1[i1]==chaine2[i2])
    {   i1++;
        i2++;
    }

    resultat = chaine1[i1]=='\0';
    return resultat;
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 17 | décomposition d'une permutation

1 Notion de permutation

Exercice 1

```
#include "permutation.h"

void saisis_permutation(int perm[N])
{  int i;
   for(i=0;i<N;i++)
   {  printf("\n Saisissez l'image de %d\n",i);
      scanf("%d",&perm[i]);
   }
}

void affiche_permutation(int perm[N])
{  int i;
   for (i=0;i<N;i++)
      printf("%3d",i);
   printf("\n");
   for (i=0;i<N;i++)
      printf("%3d",perm[i]);
   printf("\n");
}

int verifie_permutation(int perm[N])
{  int i=0,j,resultat=1;

   while(i<N-2 && resultat==1)
   {  j = i + 1;
      while(j<N-1 && resultat==1)
       {  if(perm[i]==perm[j])
             resultat=0;
          j++;
       }
      i++;
   }
   return resultat;
}
```

Exercice 2

```
#ifndef PERMUTATION_H_
#define PERMUTATION_H_

#include <stdio.h>
#include <stdlib.h>

#define N 5

void saisis_permutation(int perm[N]);
void affiche_permutation(int perm[N]);
int verifie_permutation(int perm[N]);

#endif /*PERMUTATION_H_*/
```

Exercice 3

```
#include "permutation.h"

int main()
{  int perm[N],test;
   do
   {  saisis_permutation(perm);
      test = verifie_permutation(perm);
      if(test)
         affiche_permutation(perm);
      else
         printf("Les valeurs saisies ne définissent pas une permutation.\n");
   }
   while(!test);

   return EXIT_SUCCESS;
}
```

2 Composition et inversion

Exercice 4

```
void compose_permutation(int perm1[N], int perm2[N], int resultat[N])
{  int i;
   for(i=0;i<N;i++)
      resultat[i] = perm2[perm1[i]];
}
```

Exercice 5

```
void inverse_permutation(int perm[N], int resultat[N])
{  int i;
   for(i=0; i<N;i++)
      resultat[perm[i]]=i;
}
```

Exercice 6

```
int est_identite(int perm[N])
{  int resultat=1,i=0;
   while(resultat && i<N)
   {  resultat = perm[i]==i;
      i++;
   }
   return resultat;
}
```

Exercice 7

```
int main()
{  int perm1[N]={4,3,1,2,0},perm2[N],perm3[N],test;
   inverse_permutation(perm1,perm2);
   printf("La permutation reciproque est :\n");
   affiche_permutation(perm2);
   compose_permutation(perm1,perm2,perm3);
   printf("La composee de l'originale par la reciproque est :\n");
   affiche_permutation(perm3);
   test = est_identite(perm3);
   if(test)
      printf("La composée est bien l'identite\n");
   else
      printf("La composée n'est pas l'identite\n");

   return EXIT_SUCCESS;
}
```

3 Notion de cycle

Exercice 8

```
void affiche_cycle(int perm[])
{  int i=0, suivant, debut;

   //recherche du début du cycle :
   while(perm[i]==i)
      i++;
   debut = i;
   suivant = debut;

   // affichage du cycle :
   printf("( ");
   do
   {  printf("%d ",suivant);
      suivant = perm[suivant];
   }
   while(suivant!=debut);
   printf(")");
}
```

4 Décomposition en cycles

Exercice 9

On applique l'algorithme à $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\7&1&2&4&3&5&9&0&6&8\end{pmatrix}$:

- Première itération :
  - $i_0 = 0$ car $7 \neq 0$.
  - On détecte le 2-cycle $(0\ 7)$, i.e. $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\7&1&2&3&4&5&6&0&8&9\end{pmatrix}$.
  - On prend son inverse (qui, ici, est le même cycle) $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\7&1&2&3&4&5&6&0&8&9\end{pmatrix}$.
  - On compose la permutation originale par cette réciproque, ce qui nous donne : $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&2&4&3&5&9&7&6&8\end{pmatrix}$. On recommence avec cette permutation.
- Deuxième itération :
  - $i_0 = 3$ car $4 \neq 3$.
  - On détecte le 2-cycle $(3\ 4)$, i.e. $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&2&4&3&5&6&7&8&9\end{pmatrix}$.
  - On prend son inverse (qui, ici, est le même cycle) $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&2&4&3&5&6&7&8&9\end{pmatrix}$.
  - On compose la permutation originale par cette réciproque, ce qui nous donne : $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&2&3&4&5&9&7&6&8\end{pmatrix}$. On recommence avec cette permutation.
- Troisième itération :
  - $i_0 = 6$ car $9 \neq 6$.
  - On détecte le 3-cycle $(6\ 9\ 8)$, i.e. $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&2&3&4&5&9&7&6&8\end{pmatrix}$.
  - On prend son inverse (cette fois, c'est différent) $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&2&4&3&5&\mathbf{8}&\mathbf{7}&9&\mathbf{6}\end{pmatrix}$.
  - On compose la permutation originale par cette réciproque, ce qui nous donne : $\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&2&3&4&5&6&7&8&9\end{pmatrix}$. C'est l'identité, donc l'algorithme se termine là.
- Résultat de la décomposition : $(0\ 7)(3\ 4)(6\ 9\ 8)$

Pour les 2 autres permutations proposées dans le sujet, on va trouver respectivement les décompositions (0 7 6 9 8)(3 4) et (0 1)(3 4)(6 7)(8 9).

Exercice 10

```
void decompose_permutation(int perm[N])
{  int cycle[N], copie[N], inv_cycle[N], i, debut, suivant;

   // on copie la permutation pour ne pas la modifier
   for(i=0; i<N; i++)
      copie[i] = perm[i];

   // on ne s'arrete que quand on arrive a l'identite
   while(est_identite(copie)==0)
   {  // on re-initialise le cycle
      for(i=0;i<N;i++)
         cycle[i] = i;

      // on recherche le debut du cycle
      i=0;
      while(copie[i] == i)
         i++;
      debut = i;
      suivant = debut;

      // on identifie le cycle
      do
      {  cycle[suivant] = copie[suivant];
         suivant = copie[suivant];
      }
      while(suivant != debut);

      // on affiche le cycle :
      affiche_cycle(cycle);

      // on inverse le cycle
      inverse_permutation(cycle, inv_cycle);

      // on compose avec le cycle inverse
      compose_permutation(copie,inv_cycle,copie);
   }
   printf("\n");
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 18 | nombres binaires

1 Représentation

Exercice 1

```
#define N 100
```

Exercice 2

```
void affiche_binaire(int nombre[N], int longueur)
{  // version simple
// int i;
// printf("(");
// for(i=longueur-1;i>=0;i--)
//    printf("%d",nombre[i]);
// printf(")_2");

   // version avec espace tous les 4 chiffres
   int i;
   printf("(");
   for(i=longueur-1;i>=0;i--)
   {  printf("%d",nombre[i]);
      if(i!=0 && i%4==0)
         printf(" ");
   }
   printf(")_2");
}
```

Exercice 3

```
int est_binaire(int nombre[N], int longueur)
{  int resultat=1, i=0;
   while(resultat && i<longueur)
   {  resultat = nombre[i]==0 || nombre[i]==1;
      i++;
   }
   return resultat;
}
```

2 Conversion décimal-vers-binaire

Exercice 4

```
void divise_binaire(int x, int *q, int *r)
{  *q = x / 2;
   *r = x % 2;
}
```

Exercice 5

```
void decompose_binaire(int n, int nombre[N], int *longueur)
{  int q = n;
   *longueur = 0;
   while(q!=0)
   {  divise_binaire(q, &q, &(nombre[*longueur]));
      (*longueur)++;
   }
}
```

3 Conversion binaire-vers-décimal

Exercice 6

```
int recompose_binaire(int nombre[N], int longueur)
{  int i, resultat = 0;
   for(i=longueur-1;i>=0;i--)
      resultat = resultat*2 + nombre[i];
   return resultat;
}
```

4 Opérations

Exercice 7

```
void calcule_addition(int nombre1[N], int longueur1, int nombre2[N],
                                    int longueur2, int nombre3[N], int* longueur3)
{  int i=0, retenue=0, temp, lg;

   // on determine le nombre le plus long
   if(longueur1>longueur2)
      lg = longueur1;
   else
      lg = longueur2;

   // on calcule nombre3
   while(i<=lg)
   {  temp = retenue;
      if(i<longueur1)
         temp = temp + nombre1[i];
      if(i<longueur2)
         temp = temp + nombre2[i];
      divise_binaire(temp,&retenue,&(nombre3[i]));
      i++;
   }
   // on met a jour sa longueur
   *longueur3 = i;
}
```

Exercice 8

```
int compare_binaire(int nombre1[N], int longueur1, int nombre2[N],
                                                                    int longueur2)
{  int i = 0;
   int resultat = longueur1 - longueur2;
   while(resultat==0 && i<longueur1)
   {  resultat = nombre1[i] - nombre2[i];
      i++;
   }
   return resultat;
}
```

5 Exponentiation

Exercice 9

```
int calcule_puissance1(float x, int n, float *res)
{  int i;
   *res = 1;
   for(i=0;i<n;i++)
      *res = *res * x;
   return i;
}
```

Exercice 10

```
int calcule_puissance2(float x, int n, float *res)
{  int nombre[N], longueur, i;
```

```
        float temp = x;

        // on initialise le resultat
        *res = 1;

        // on decompose n
        decompose_binaire(n,nombre,&longueur);

        // on applique le principe explique dans le sujet
        for(i=0;i<longueur;i++)
        {   if(nombre[i])
               *res = *res * temp;
            temp = temp * temp;
        }
        return i;
    }
```

Exercice 11

Le nombre d'itérations pour calculer x^n à l'aide de la fonction `calcule_puissance1` est égal à n, car on décompose x^n en un produit de n termes $x \times x \ldots \times x$.

Pour `calcule_puissance2`, le nombre d'itérations correspond à la longueur de la décomposition binaire de n, c'est-à-dire qu'il est inférieur à $\log_2(n)$. Comme $\lim_{\infty} \frac{\log_2(n)}{n} = 0$, pour les grandes valeur de n, cet algorithme est beaucoup plus efficace que l'algorithme classique, d'où son nom d'algorithme d'*exponentiation rapide*.

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 19 | algorithme de Johnson

2 Implémentation

Exercice 1

$\overleftarrow{0}\,\overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{3}$	$\overrightarrow{3}\,\overleftarrow{0}\,\overleftarrow{2}\,\overleftarrow{1}$	$\overleftarrow{2}\,\overleftarrow{0}\,\overleftarrow{1}\,\overleftarrow{3}$	$\overrightarrow{3}\,\overrightarrow{2}\,\overleftarrow{1}\,\overleftarrow{0}$	$\overleftarrow{1}\,\overrightarrow{2}\,\overleftarrow{0}\,\overleftarrow{3}$	$\overrightarrow{3}\,\overleftarrow{1}\,\overleftarrow{0}\,\overrightarrow{2}$
$\overleftarrow{0}\,\overleftarrow{1}\,\overleftarrow{3}\,\overleftarrow{2}$	$\overleftarrow{0}\,\overrightarrow{3}\,\overleftarrow{2}\,\overleftarrow{1}$	$\overleftarrow{2}\,\overleftarrow{0}\,\overrightarrow{3}\,\overleftarrow{1}$	$\overrightarrow{2}\,\overrightarrow{3}\,\overleftarrow{1}\,\overleftarrow{0}$	$\overleftarrow{1}\,\overrightarrow{2}\,\overrightarrow{3}\,\overleftarrow{0}$	$\overleftarrow{1}\,\overrightarrow{3}\,\overleftarrow{0}\,\overrightarrow{2}$
$\overleftarrow{0}\,\overrightarrow{3}\,\overleftarrow{1}\,\overleftarrow{2}$	$\overleftarrow{0}\,\overrightarrow{2}\,\overrightarrow{3}\,\overleftarrow{1}$	$\overleftarrow{2}\,\overrightarrow{3}\,\overleftarrow{0}\,\overleftarrow{1}$	$\overrightarrow{2}\,\overleftarrow{1}\,\overrightarrow{3}\,\overleftarrow{0}$	$\overleftarrow{1}\,\overrightarrow{3}\,\overrightarrow{2}\,\overleftarrow{0}$	$\overleftarrow{1}\,\overleftarrow{0}\,\overrightarrow{3}\,\overrightarrow{2}$
$\overrightarrow{3}\,\overleftarrow{0}\,\overleftarrow{1}\,\overleftarrow{2}$	$\overleftarrow{0}\,\overrightarrow{2}\,\overleftarrow{1}\,\overrightarrow{3}$	$\overrightarrow{3}\,\overrightarrow{2}\,\overleftarrow{0}\,\overleftarrow{1}$	$\overrightarrow{2}\,\overleftarrow{1}\,\overleftarrow{0}\,\overrightarrow{3}$	$\overrightarrow{3}\,\overleftarrow{1}\,\overrightarrow{2}\,\overleftarrow{0}$	$\overleftarrow{1}\,\overleftarrow{0}\,\overrightarrow{2}\,\overrightarrow{3}$

Exercice 2

```
typedef enum
{  gauche = -1,
   droite = 1
} t_girouette;

typedef struct
{  int val;
   t_girouette gir;
} t_entier_girouette;
```

Exercice 3

```
void initialise_permutation(t_entier_girouette permutation[], int n)
{  int i;
   for(i=0;i<n;i++)
   {  permutation[i].val = i;
      permutation[i].gir = gauche;
   }
}
```

Exercice 4

```
void affiche_permutation(t_entier_girouette permutation[], int n)
{  int i;
   for (i=0;i<n;i++)
      if(permutation[i].gir==gauche)
         printf(" <- ");
      else
         printf(" -> ");
   printf("\n");
   for(i=0;i<n;i++)
      printf(" %2d ", permutation[i].val);
}
```

Exercice 5

```
void identifie_pgem(t_entier_girouette permutation[], int n, int *val, int
*position)
{  int girouette, i, j;
   *position=-1;
   *val=-1;

   for(i=0;i<n;i++)
   {  // on ne considère que les vals pouvant devenir le P.G.M.
      if(permutation[i].val > *val)
      {  girouette = permutation[i].gir;
```

```
            j = i + girouette;
            // on détermine si l'entier est mobile
            if(j>=0 && j<n && permutation[i].val>permutation[j].val)
            {   *position = i;
                *val = permutation[i].val;
            }
        }
    }
}
```

Exercice 6

```
void deplace_entier(t_entier_girouette permutation[], int position)
{   t_entier_girouette temp;
    int position2 ;
    position2 = position + permutation[position].gir;
    temp = permutation[position];
    permutation[position] = permutation[position2];
    permutation[position2] = temp;
}
```

Exercice 7

```
void inverse_girouette(t_entier_girouette permutation[], int n, int m)
{   int i;

    for(i=0;i<n;i++)
    {   if(permutation[i].val > m)
        {   if(permutation[i].gir == gauche)
                permutation[i].gir = droite;
            else
                permutation[i].gir = gauche;
        }
    }
}
```

Exercice 8

```
void johnson(int n)
{   t_entier_girouette permutation[n+2];
    int position, val;

    // point 1 de l'algo du sujet
    initialise_permutation(permutation,n);
    // point 2
    do
    {   // point 3
        affiche_permutation(permutation,n);
        printf("\n");
        // point 4
        identifie_pgem(permutation,n,&val,&position);
        // point 5
        if(position!=-1)
        {   // point 6
            deplace_entier(permutation,position);
            // point 7
            inverse_girouette(permutation,n,val);
        }
    }
    // points 8 et 9
    while(position!=-1);
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 20 | manipulation de dates

1 Représentation des mois

Exercice 2

```
typedef enum
{  jan=1,
   fev,mars,avr,mai,juin,juil,aout,sept,oct,nov,dec
} t_mois;
```

Exercice 3

```
void affiche_mois(t_mois m)
{  switch(m)
   {  case jan:
         printf("janvier");
         break;
      case fev:
         printf("fevrier");
         break;
      case mars:
         printf("mars");
         break;
      case avr:
         printf("avril");
         break;
      case mai:
         printf("mai");
         break;
      case juin:
         printf("juin");
         break;
      case juil:
         printf("juillet");
         break;
      case aout:
         printf("aout");
         break;
      case sept:
         printf("septembre");
         break;
      case oct:
         printf("octobre");
         break;
      case nov:
         printf("novembre");
         break;
      case dec:
         printf("decembre");
         break;
   }
}
```

2 Représentation des dates

Exercice 4

```
typedef struct
{  int jour;
   t_mois mois;
   int annee;
} t_date;
```

Exercice 5

```
void affiche_date(t_date date)
{  printf("%d ",date.jour);
   affiche_mois(date.mois);
   printf(" %d",date.annee);
}
```

Exercice 6

```
void saisis_date(t_date *date)
{  int temp;

   // jour
   printf("Entrez le jour : ");
   scanf("%d",&temp);
   date->jour = temp;

   // mois
   printf("Entrez le mois : ");
   scanf("%d",&temp);
   date->mois = (t_mois)temp;

   // annee
   printf("Entrez l'annee : ");
   scanf("%d",&temp);
   date->annee = temp;
}
```

Exercice 7

```
int compare_dates(t_date date1, t_date date2)
{  int resultat = date1.annee - date2.annee;
   if(resultat==0)
   {  resultat = date1.mois - date2.mois;
      if(resultat==0)
         resultat = date1.jour - date2.jour;
   }
   return resultat;
}
```

Exercice 8

```
t_date copie_date(t_date date)
{  t_date resultat = {date.jour,date.mois,date.annee};
   return resultat;
}
```

3 Calcul sur les années

Exercice 9

```
int est_bissextile(int annee)
{  int resultat;
   if(annee%4==0 && annee%100!=0)
      resultat = 1;
   else
      resultat = annee%400==0;
   return resultat;
}
```

Exercice 10

```
int indique_jours(int annee, t_mois mois)
```

```
{  int resultat;
   switch(mois)
   {  case jan:
       case mars:
       case mai:
       case juil:
       case aout:
       case oct:
       case dec:
          resultat = 31;
          break;
      case avr:
      case juin:
      case sept:
      case nov:
          resultat = 30;
          break;
      default: //fev
          if(est_bissextile(annee))
             resultat = 29;
          else
             resultat = 28;
   }
   return resultat;
}
```

Exercice 11

```
void saisis_date_verif(t_date *date)
{  int temp,duree;
   int limite = 2014;

   // annee
   do
   {  printf("Entrez l'annee (1900<=entier<=%d) : ",limite);
      scanf("%d",&temp);
   }
   while(temp<1900 || temp>limite);
   date->annee = temp;

   // mois
   do
   {  printf("Entrez le mois (1<=entier<=12) : ");
      scanf("%d",&temp);
   }
   while(temp<1 || temp>12);
   date->mois = (t_mois)temp;

   // jour
   duree = indique_jours(date->annee,date->mois);
   do
   {  printf("Entrez le jour (1<=entier<=%d) : ",duree);
      scanf("%d",&temp);
   }
   while(temp<1 || temp>duree);
   date->jour = temp;
}
```

4 Calculs divers

Exercice 12

```
void ajoute_jour(t_date* date)
{  int duree = indique_jours(date->annee, date->mois);
   if(date->jour<duree)
      date->jour = date->jour + 1;
   else
   {  date->jour = 1;
```

```
        if(date->mois==dec)
        {  date->mois = jan;
           date->annee = date->annee + 1;
        }
        else
           date->mois = date->mois + 1;
    }
}
```

Exercice 13

```
int calcule_duree(t_date date1, t_date date2)
{  t_date avant,apres;
   int compare = compare_dates(date1,date2);
   int resultat=0, inverse;

   // initialisation
   if(compare<0)
   {  avant = copie_date(date1);
      apres = copie_date(date2);
      inverse = 0;
   }
   else
   {  avant = copie_date(date2);
      apres = copie_date(date1);
      inverse = 1;
   }

   // calcul
   while(compare!=0)
   {  ajoute_jour(&avant);
      resultat++;
      compare = compare_dates(avant,apres);
   }

   // oppose ?
   if(inverse)
      resultat = -resultat;

   return resultat;
}
```

Exercice 14

```
void saisis_recettes(int recettes[])
{  t_mois mois;
   for(mois=jan;mois<=dec;mois++)
   {  printf("Entrez la recettes du mois de ");
      affiche_mois(mois);
      printf(" : ");
      scanf("%d",&recettes[mois-1]);
   }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 21 | carrés latins

1 Définition

Exercice 1

Un carré latin d'ordre 3 :

$$\begin{bmatrix} 0 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 2 \end{bmatrix}$$

Un carré latin d'ordre 5 :

$$\begin{bmatrix} 0 & 2 & 1 & 4 & 3 \\ 2 & 1 & 0 & 3 & 4 \\ 3 & 4 & 2 & 1 & 0 \\ 4 & 0 & 3 & 2 & 1 \\ 1 & 3 & 4 & 0 & 2 \end{bmatrix}$$

Exercice 2

```
void affiche_cl(int m[N][N])
{  int i, j;
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         printf("% 4d",m[i][j]);
      printf("\n");
   }
}
```

Exercice 3

```
void genere_cl_permutation(int m[N][N])
{  int i, j;
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         m[i][j] = (i+j) % N;
   }
}
```

2 Ronds & croix

Exercice 4

```
typedef enum
{  croix,
   rond
} t_symbole;
```

Exercice 5

```
void initialise_ronds_vecteur(t_symbole tab[N])
{  int i;
   for(i=0;i<N;i++)
      tab[i] = rond;
}
```

Exercice 6

```
int compte_ronds_vecteur(t_symbole tab[N])
{  int resultat=0, i;
   for(i=0; i<N; i++)
      if(tab[i]==rond)
         resultat++;
   return resultat;
}
```

Exercice 7

```
int contient_rond_vecteur(t_symbole tab[N])
{  int nb_ronds = compte_ronds_vecteur(tab);
   int resultat = nb_ronds >= 1;
   return resultat;
}
```

3 Test de propriété

Exercice 8

```
int teste_vecteur(int m[N][N], int pos, int ligne)
{  int i, resultat;

   // on initialise le tableau avec des ronds
   t_symbole tab[N];
   initialise_ronds_vecteur(tab);

   // on rajoute les croix
   for(i=0;i<N;i++)
   {  if(ligne)
         tab[m[pos][i]] = croix;
      else
         tab[m[i][pos]] = croix;
   }

   // on teste la presence de ronds
   resultat = !contient_rond_vecteur(tab);
   return resultat;
}
```

Exercice 9

```
int est_carre_latin(int m[N][N])
{  int pos,ligne=0,resultat=1;

   // on teste les colonnes puis les lignes
   while(resultat && ligne<2)
   {  pos = 0;
      while(resultat && pos<N)
      {  resultat = teste_vecteur(m,pos,ligne);
         pos++;
      }
      ligne++;
   }

   return resultat;
}
```

4 Orthogonalité

Exercice 10

```
void initialise_ronds_matrice(t_symbole m[N][N])
{  int i;
   for(i=0;i<N;i++)
      initialise_ronds_vecteur(m[i]);
}
```

```
int compte_ronds_matrice(t_symbole m[N][N])
{  int resultat=0, i, temp;
   for(i=0; i<N; i++)
   {  temp = compte_ronds_vecteur(m[i]);
      resultat = resultat + temp;
   }
   return resultat;
}

int contient_rond_matrice(t_symbole m[N][N])
{  int nb_ronds = compte_ronds_matrice(m);
   int resultat = nb_ronds >= 1;
   return resultat;
}
```

Exercice 11

```
int sont_orthogonaux(int a[N][N], int b[N][N])
{  int i=0,j,resultat=1;

   // on initialise la matrice avec des ronds
   t_symbole m[N][N];
   initialise_ronds_matrice(m);

   // on rajoute les croix
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
         m[a[i][j]][b[i][j]] = croix;
   }

   // on teste la presence de ronds
   resultat = !contient_rond_matrice(m);

   return resultat;
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 22 | représentation d'une promotion

1 Représentation d'un étudiant

Exercice 2

```
typedef enum
{  feminin,
   masculin
} t_genre;
```

Exercice 3

```
typedef struct
{  char nom[TAILLE_MAX_NOM];
   char prenom[TAILLE_MAX_NOM];
   t_date naissance;
   t_genre genre;
} t_etudiant;
```

Exercice 4

```
void saisis_etudiant(t_etudiant* e)
{  printf ("Entrez le prenom : ");
   scanf("%s",e->prenom);
   printf ("Entrez le nom : ");
   scanf("%s",e->nom);
   printf("Date de naissance : ");
   saisis_date(&e->naissance);
   printf ("Entrez le genre : ");
   scanf("%d",&(e->genre));
}
```

Exercice 5

```
void affiche_etudiant(t_etudiant e)
{  if(e.genre)
      printf("%s %s, ne le ",e.prenom, e.nom);
   else
      printf("%s %s, nee le ",e.prenom, e.nom);
   affiche_date(e.naissance);
}
```

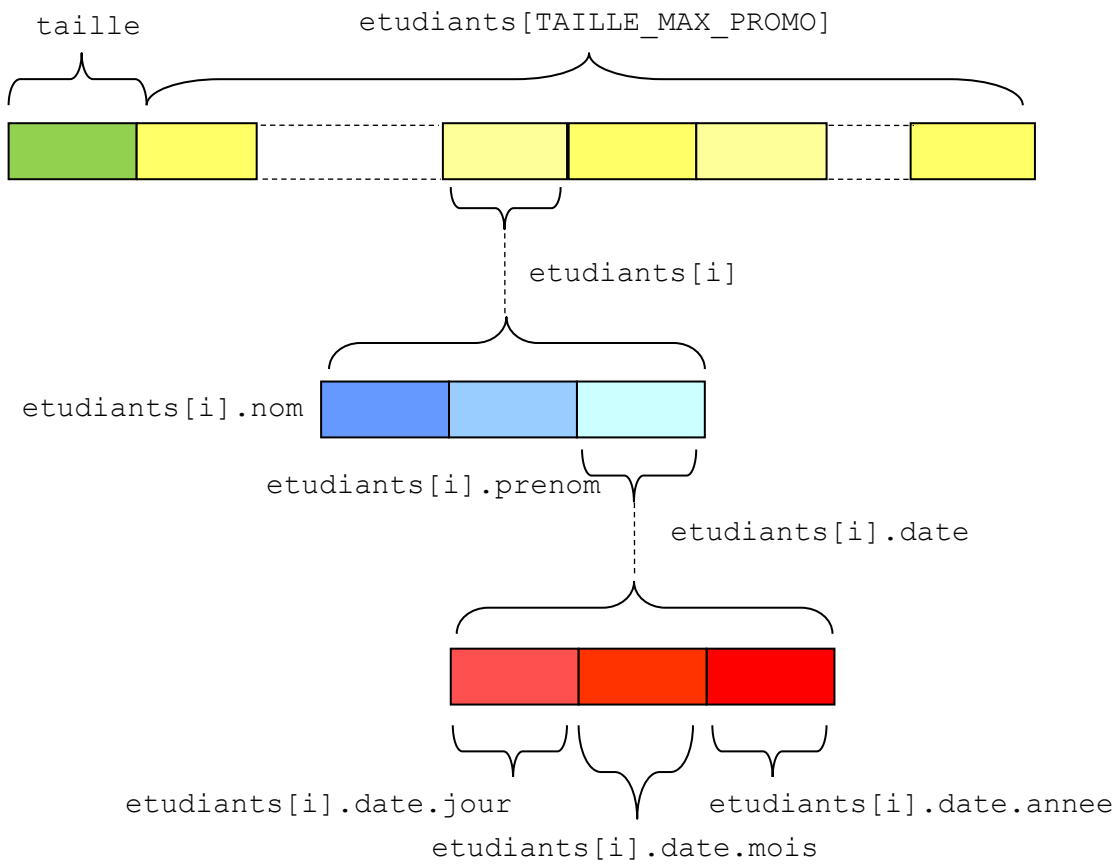
2 Représentation d'une promotion

Exercice 6

- Type `t_promo` :

```
typedef struct
{  int taille;
   t_etudiant etudiants[TAILLE_MAX_PROMO];
} t_promo;
```

- Représentation graphique d'une promotion :



Exercice 7

```
void saisis_promotion(t_promo* p)
{  char reponse;
   p->taille = 0;
   do
   {  printf("-- Etudiant n.%d --\n",p->taille);
      saisis_etudiant(&(p->etudiants[p->taille]));
      printf("Voulez-vous saisir un autre etudiant (O/N) ? ");
      p->taille++;
      scanf("%c",&reponse);
   }
   while(reponse!='n' && reponse!='N');
   printf("%d etudiants ont ete saisis\n",p->taille);
 }
```

Exercice 8

```
void affiche_promotion(t_promo p)
{  int i;
   for(i=0;i<p.taille;i++)
   {  printf("%d. ",i);
      affiche_etudiant(p.etudiants[i]);
      printf("\n");
   }
}
```

3 Recherche dans la promotion

Exercice 9

```
t_promo initialise_promotion()
{  t_promo resultat = {9,
   {  {"Emreoglu", "Emre", {20, 11, 1985}, 1},
      {"Ozturk", "Zeynep", {8,8,1988}, 0},
```

```
        {"Ozanoglu", "Ozan", {10, 10, 1986}, 1},
        {"Canoglu", "Can", {8, 8, 1984}, 1},
        {"Kucuk", "Naz", {2, 2, 1985}, 0},
        {"Kucukkarakurt", "Dilek", {7, 7, 1985}, 0},
        {"Onay", "Sevgi", {4, 4, 1985}, 0},
        {"Kocak", "Ozgur", {6, 5, 1985}, 1},
        {"Kucuk", "Hakan", {3, 3, 1986}, 1}
    }};
    return resultat;
}
```

Exercice 10

```
void cherche_homonymes(t_promo p, char* nom)
{  int i;
   printf("Résultat de la recherche pour la chaîne \"%s\" :\n",nom);
   for(i=0;i<p.taille;i++)
   {  if(compare_chaines(p.etudiants[i].nom,nom)==0)
      {  printf("- ");
         affiche_etudiant(p.etudiants[i]);
         printf("\n");
      }
   }
}
```

Exercice 11

```
void cherche_prefixe(t_promo p, t_promo* res, char *prefixe)
{  int i,j=0;
   for(i=0;i<p.taille;i++)
   {  if(est_prefixe(prefixe,p.etudiants[i].nom))
      {  res->etudiants[j] = p.etudiants[i];
         res->taille ++;
         j++;
      }
   }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 23 | partition d'un entier

1 Définition et représentation

Exercice 1

```
#define L_MAX 100

typedef struct
{  int n;
   int u[L_MAX];
}t_partition;
```

Exercice 2

```
void affiche_partition(t_partition p)
{  int i = 0;
   printf("%d = (",p.n);
   while(p.u[i]!=0)
   {  printf(" %d",p.u[i]);
      i++;
   }
   printf(" )\n");
}
```

Exercice 3

```
void copie_partition(t_partition p, t_partition *copie)
{  int i;
   copie->n = p.n;

   i = 0;
   while(p.u[i] !=0)
   {  copie->u[i] = p.u[i];
      i++;
   }
   copie->u[i] = 0;
}
```

2 Comparaison

Exercice 4

```
int compare_partitions(t_partition p1, t_partition p2)
{  int resultat;
   int i = 0;
   while(p1.u[i]==p2.u[i] && p1.u[i]!=0)
      i++;
   resultat = p1.u[i]-p2.u[i];
   return resultat;
}
```

3 Partitions triviales

Exercice 5

```
void calcule_partition_courte(int n, t_partition *p)
{  p->n = n;
```

```
    p->u[0] = n;
    p->u[1] = 0;
}
```

Exercice 6

```
void calcule_partition_longue(int n, t_partition *p)
{  int i;
   p->n = n;
   for(i=0;i<n;i++)
      p->u[i] = 1;
   p->u[i] = 0;
}
```

4 Génération

Exercice 7

On applique l'algorithme, aux partitions de 12 spécifiées dans le sujet :

- $u = (12,0)$:
  - $i_0 = 0$, donc $v_0 = 12 - 1 = 11$.
  - $a = 0$ et $a + 1 = 1 = 11 \times 0 + 1$, donc $v_1 = r = 1$.
  - Donc : $v = (11,1,0)$
- $u = (5,5,1,1,0)$.
  - $i_0 = 1$, donc $v_0 = 5$ et $v_1 = 5 - 1 = 4$.
  - $a = 2$ et $a + 1 = 3 = 4 \times 0 + 3$, donc $v_2 = r = 3$.
  - Donc : $v = (5,4,3,0)$
- $u = (1,1,1,1,1,1,1,1,1,1,1,0)$.
  - On ne peut pas calculer i_0 donc l'algorithme s'arrête (u est la première partition de 12 pour l'ordre lexicographique).

Exercice 8

```
void calcule_partition_precedente(t_partition p0, t_partition* p1)
{  int i, j, a, q, r;

   // initialisation de n dans p1
   p1->n = p0.n;

   // copie des termes >1
   i = 0;
   while(p0.u[i+1]>1)
   {  p1->u[i] = p0.u[i];
      i++;
   }
   j = i;

   // traitement du jeme terme
   p1->u[j] = p0.u[j] - 1;

   // on compte les 1 dans p0
   i = j + 1;
   a = 0;
   while(p0.u[i]==1)
   {  a++;
      i++;
   }

   // on calcule le quotient et le reste
   q = (a+1) / p1->u[j];
   r = (a+1) % p1->u[j];

   // on rajoute les 1 manquants
   for(i=j+1;i<=j+q;i++)
```

```
            p1->u[i] = p1->u[j];
        if(r != 0)
        {   p1->u[i] = r;
            i++;
        }

        // on place le zero a la fin
        p1->u[i] = 0;
    }
```

Exercice 9

```
    void affiche_toutes_partitions(int n)
    {   t_partition p0, p1, p2;
        calcule_partition_courte(n, &p0);
        calcule_partition_longue(n, &p2);
        affiche_partition(p0);

        while(compare_partitions(p0,p2)>0)
        {   calcule_partition_precedente(p0, &p1);
            affiche_partition(p1);
            copie_partition(p1,&p0);
        }
    }
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 24 | rotation d'un carré

1 Matrice de rotation

Exercice 1

```
void affiche_matrice(double mat[2][2])
{  int i,j;
   for(i=0;i<2;i++)
   {  printf("(");
      for(j=0;j<2;j++)
         printf(" % 2lf",mat[i][j]);
      printf(" )\n");
   }
}
```

Exercice 2

```
void initialise_rotation(double m[2][2], double theta)
{  m[0][0] = cos(theta);
   m[0][1] = -sin(theta);
   m[1][0] = -m[0][1];
   m[1][1] = m[0][0];
}
```

2 Rotation d'un vecteur

Exercice 3

```
struct s_vecteur
{  double x;
   double y;
};
```

Exercice 4

```
void affiche_vecteur(struct s_vecteur v)
{  printf("( % 2lf )\n( % 2lf )\n",v.x,v.y);
}
```

Exercice 5

```
void applique_rotation(double m[2][2], struct s_vecteur v,
                                                  struct s_vecteur* resultat)
{  resultat->x = m[0][0]*(v.x)+m[0][1]*(v.y);
   resultat->y = m[1][0]*(v.x)+m[1][1]*(v.y);
}
```

3 Rotation d'un carré

Exercice 6

```
void dessine_carre(int x_O, int y_O, struct s_vecteur v_OA, Uint32 coul)
{  int x_A,y_A, x_B, y_B,x_C,y_C,x_D,y_D;
   x_A = x_O + (int)v_OA.x;
   y_A = y_O + (int)v_OA.y;
   x_B = x_O - (int)v_OA.y;
   y_B = y_O + (int)v_OA.x;
```

```
    x_C = x_O - (int)v_OA.x;
    y_C = y_O - (int)v_OA.y;;
    x_D = x_O + (int)v_OA.y;
    y_D = y_O - (int)v_OA.x;;
    dessine_segment(x_A,y_A,x_B,y_B,coul);
    dessine_segment(x_B,y_B,x_C,y_C,coul);
    dessine_segment(x_C,y_C,x_D,y_D,coul);
    dessine_segment(x_D,y_D,x_A,y_A,coul);
}
```

Exercice 7

```
void tourne_carre(int x_O, int y_O, struct s_vecteur v_OA, Uint32 coul, int k)
{  int i;
   double m[2][2];

   // on initialise la matrice de rotation
   initialise_rotation(m,2*PI/k);

   // on applique iterativement la rotation
   for(i=0;i<k;i++)
   {  // on dessine le carre en couleur
      dessine_carre(x_O, y_O, v_OA, coul);
      rafraichis_fenetre();
      // on attend un peu
      SDL_Delay(DELAI);
      // on efface le carre en dessinant en noir
      dessine_carre(x_O, y_O, v_OA, C_NOIR);
      rafraichis_fenetre();
      // on applique la rotation pour la prochaine iteration
      applique_rotation(m, v_OA, &v_OA);
   }
}
```

Exercice 8

```
void tourne_carre2(int x_O, int y_O, struct s_vecteur v_OA, int k)
{  int i;
   double m[2][2];
   Uint8 r=255,v=0,b=0;
   Uint32 coul;

   // on initialise la matrice de rotation
   initialise_rotation(m,2*PI/k);

   // on applique iterativement la rotation
   for(i=0;i<k/4;i++)
   {  // on dessine le carre en couleur
      coul = convertis_rvb(r, v, b);
      dessine_carre(x_O, y_O, v_OA, coul);
      rafraichis_fenetre();
      // on attend un peu
      SDL_Delay(DELAI);
      // on applique la rotation pour la prochaine iteration
      applique_rotation(m, v_OA, &v_OA);
      // on calcule la nouvelle couleur
      v = v + 255/(k/4);
      b = v;
   }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 25 | zoom d'une image

1 Présentation

Exercice 1

```
void trace_surface(SDL_Surface* surface)
{  efface_fenetre();
   dessine_surface(0,0,surface);
   rafraichis_fenetre();
   attends_touche();
}
```

Exercice 2

```
   // surfaces utilisee pour representer la photo
   SDL_Surface *photo, *zoomout, *zoomin, *zoomoutin;

   // initialisation de la SDL
   initialise_fenetre("Zoom d'une image");

   // chargement et affichage d'une photo
   photo = charge_image("album/image2.bmp");
   trace_surface(photo);
```

2 Agrandissement

Exercice 3

```
SDL_Surface* agrandis(SDL_Surface* source, int x, int y, int largeur,
                                                          int hauteur, int coef)
{  // coordonnees du pixel courant de la zone d'interet
   int i,j;
   // coordonnees du pixel courant de l'agrandissement
   int m,n;
   // index utilises pour dessiner des carres de plusieurs pixels
   int k,l;
   // couleur du pixel courant de la fenetre
   Uint32 coul;
   SDL_Surface* destination = initialise_surface(largeur*coef,hauteur*coef);

   // parcourt chaque pixel de cette zone
   for(i=x;i<x+largeur;i++)
   {  m = (i-x)*coef;
      for(j=y;j<y+hauteur;j++)
      {  n = (j-y)*coef;
         // on recupere la couleur du pixel original
         coul = couleur_pixel_surface(source,i,j);
         // on dessine un carre de la meme couleur
         for(k=0;k<coef;k++)
         {  for(l=0;l<coef;l++)
            {  allume_pixel_surface(destination, m+k, n+l, coul);
            }
         }
      }
   }

   return destination;
```

```
}
```

Exercice 4

```
// agrandissement et enregistrement de la photo
zoomin = agrandis(photo, 70, 70, 200, 150, 4);
sauve_surface(zoomin,"album/image2.zoomin.bmp");
dessine_rectangle(70,70,200,150,C_BLANC);
rafraichis_fenetre();
attends_touche();
trace_surface(zoomin);
```

3 Réduction

Exercice 5

```
SDL_Surface* reduis(SDL_Surface* source, int coef)
{  // indices des pixels
   int i,j,m,n;
   // composantes de couleur
   int r,v,b;
   Uint8 rouge,vert,bleu;
   Uint32 coul;

   // surface contenant la version reduite
   int largeur = source->w / coef;
   int hauteur = source->h / coef;
   SDL_Surface* destination = initialise_surface(largeur,hauteur);

   // on dessine les pixels moyens dans la nouvelle surface
   for(m=0;m<largeur;m++)
   {  for(n=0;n<hauteur;n++)
      {  r=0; v=0; b=0;
         for(i=m*coef;i<(m+1)*coef;i++)
         {  for(j=n*coef;j<(n+1)*coef;j++)
            {  composantes_pixel_surface(source,i,j,&rouge,&vert,&bleu);
               r=r+rouge; v=v+vert; b=b+bleu;
            }
         }
         r=r/(coef*coef); v=v/(coef*coef); b=b/(coef*coef);
         coul = convertis_rvb(r, v, b);
         allume_pixel_surface(destination, m, n, coul);
      }
   }

   return destination;
}
```

Exercice 6

```
// on re-affiche l'image originale
trace_surface(photo);

// reduction de la zone d'interet
zoomout = reduis(photo, 2);
sauve_surface(zoomout,"album/image2.zoomout.bmp");
trace_surface(zoomout);
```

Exercice 7

```
SDL_Surface* reduis_agrandis(SDL_Surface* source, int coef)
{  // centre de l'image originale
   int largeur = source->w/coef;
   int hauteur = source->h/coef;

   // on reduit l'image
   SDL_Surface* zoomout = reduis(source,coef);

   // on agrandit l'image reduite
```

```
        SDL_Surface* zoomin = agrandis(zoomout,0,0,largeur,hauteur,coef);

        return zoomin;
    }
```

Exercice 8

```
        // on re-affiche l'image originale
        trace_surface(photo);

        // reduction puis agrandissement
        zoomoutin = reduis_agrandis(photo, 8);
        sauve_surface(zoomoutin,"album/image2.zoomoutin.bmp");
        trace_surface(zoomoutin);
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 26 | automates finis

3 Implémentation

Exercice 1

```
#define LONGUEUR_MAX_MOT 100
#define NB_MAX_ETATS 5
#define TAILLE_ALPHABET 2
```

Exercice 2

```
typedef struct
{  int nb_etats;
   int etat_initial;
   int etat_acceptant;
   int matrice_transition[NB_MAX_ETATS][TAILLE_ALPHABET];
} t_automate;
```

Exercice 3

```
void affiche_automate(t_automate a)
{  int i,j;
   printf("Le nombre d'etats de l'automate est %d\n", a.nb_etats);
   printf("L'etat initial est E%d\n", a.etat_initial);
   printf("L'etat acceptant est E%d\n", a.etat_acceptant);
   printf("La matrice de transition est :\n ");
   for(i=0;i<TAILLE_ALPHABET;i++)
      printf("\t%c",'a'+i);
   printf("\n");
   for(i=0;i<a.nb_etats;i++)
   {  printf("E%d",i);
      for(j=0;j<TAILLE_ALPHABET;j++)
         printf("\tE%d", a.matrice_transition[i][j]);
      printf("\n");
   }
}
```

Exercice 4

```
void saisis_automate(t_automate* a)
{  int i, j;

   printf("Entrez le nombre d'etats : ");
   scanf("%d",&a->nb_etats);
   printf("Entrez l'etat initial : ");
   scanf("E%d",&a->etat_initial);
   printf("Entrez l'etat acceptant : ");
   scanf("E%d",&a->etat_acceptant);
   for(i=0;i<a->nb_etats;i++)
   {  printf("- Entrez les transitions de l'etat E%d :\n",i);
      for(j=0;j<TAILLE_ALPHABET;j++)
      {  printf("  %c : ",'a'+j);
         scanf("%d", &a->matrice_transition[i][j]);
      }
   }
}
```

Exercice 5

```
int applique_transition(t_automate a, int etat_courant, char lettre)
{  int resultat = a.matrice_transition[etat_courant][lettre-'a'];
   return resultat;
}
```

Exercice 6

- Fonction `teste_mot` :

```
void teste_mot(t_automate a)
{  char mot[LONGUEUR_MAX_MOT];
   char *lettre_courante;
   int etat_courant;

   // on affiche l'automate recu
   affiche_automate(a);

   // on saisit le mot
   printf("Entrez un mot contenant uniquement les lettres a et b : ");
   scanf("%s",mot);

   // on applique l'automate
   lettre_courante = mot;
   etat_courant = a.etat_initial;
   while(*lettre_courante != '\0')
   {  etat_courant = applique_transition(a,etat_courant,*lettre_courante);
      lettre_courante++;
   }

   // on affiche le resultat
   if(etat_courant==a.etat_acceptant)
      printf("Le mot \"%s\" est accepte par cet automate\n",mot);
   else
      printf("Le mot \"%s\" est refuse par cet automate\n",mot);
}
```

- Fonction `main` :

```
int main(int argc, char *argv[])
{  t_automate a = {4,0,2,{{1,3},{2,1},{2,1},{3,3}}};
   teste_mot(a);

   return EXIT_SUCCESS;
}
```

Exercice 7

```
int main(int argc, char *argv[])
{  t_automate a = {5,0,4,{{1,0},{3,2},{4,2},{3,3},{3,4}}};
   teste_mot(a);

   return EXIT_SUCCESS;
}
```

Une analyse informelle du graphe de l'automate proposé révèle que celui-ci accepte les mots contenant *exactement* 2 fois la lettre a, ces occurrences devant être séparées par une ou plusieurs occurrences de la lettre b.

- L'état initial E_0 permet d'insérer autant de b que l'on veut au début du mot (y compris ne pas insérer de b du tout). On ne passe en E_1 que quand l'on rencontre un a. Une chaîne sans aucun a sera donc refusée, car E_0 n'est pas un état acceptant.
- Dans l'état E_1, une lettre a amène dans l'état E_3. On ne peut pas sortir de E_3, et ce n'est pas un état acceptant, donc un mot qui amène dans cet état sera forcément refusé. Ici, cela signifie que 2 lettres *a consécutives* entrainent un refus du mot.
- Au contraire, une lettre b rencontrée en E_1 amène en E_2. Seul un autre a permet alors de passer en E_4, qui est l'état acceptant. Cela signifie qu'il est nécessaire

d'avoir un second a dans le mot, et qu'il peut être séparé du premier par autant de b que l'on veut.

- En E_4, tout a rencontré amène à l'état E_3, et on a déjà vu que celui-ci entraîne un échec. Cela signifie donc qu'il n'est pas permis d'avoir un $3^{\text{ème}}$ a dans le mot. Au contraire, les b placés en fin de mot sont autorisés, puisqu'on reste en E_4 quand on rencontre cette lettre.

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 27 | championnat de football

1 Équipes

Exercice 1

```
typedef enum
{  akh, ant, bes, bur, riz, ela,
   esk, fen, gal, gaz, ank, kar,
   kas, erc, kay, kon, siv, tra
} t_code;
```

Exercice 2

```
typedef struct
{  t_code code;
   char *nom;
   char *ville;
} t_equipe;
```

Exercice 3

```
t_equipe equipes[N]=
{  {akh,"Akhisar","Akhisar"},
   {ant,"Antalyaspor","Antalya"},
   {bes,"Besiktas","Istanbul"},
   {bur,"Bursaspor","Bursa"},
   {riz,"Rizespor","Rize"},
   {ela,"Elazigspor","Elazig"},
   {esk,"Eskisehirspor","Eskisehir"},
   {fen,"Fenerbahce","Istanbul"},
   {gal,"Galatasaray","Istanbul"},
   {gaz,"Gaziantepspor","Gaziantep"},
   {ank,"Genclerbirligi","Ankara"},
   {kar,"Karabukspor","Karabuk"},
   {kas,"Kasimpasa","Istanbul"},
   {erc,"Erciyesspor","Kayseri"},
   {kay,"Kayserispor","Kayseri"},
   {kon,"Konyaspor","Konya"},
   {siv,"Sivasspor","Sivas"},
   {tra,"Trabzonspor","Trabzon"}
};
```

Exercice 4

```
void affiche_nom(t_code code)
{  t_equipe equipe = equipes[code];
   printf("%s",equipe.nom);
}
```

2 Rencontres

Exercice 5

```
typedef struct
{  t_code code_local;
   t_code code_visiteur;
   int buts_local;
   int buts_visiteur;
```

```
} t_rencontre;
```

Exercice 6

```
void affiche_rencontre(t_rencontre rencontre)
{  char *nom_local = equipes[rencontre.code_local].nom;
   char *nom_visiteur = equipes[rencontre.code_visiteur].nom;
   printf("%s                    %d                    -                    %d
%s",nom_local,rencontre.buts_local,rencontre.buts_visiteur,nom_visiteur);
}
```

Exercice 7

```
void initialise_score(t_rencontre* rencontre)
{  rencontre->buts_local = rand()%6;
   rencontre->buts_visiteur = rand()%6;
}
```

3 Championnat

Exercice 8

```
void initialise_championnat(t_rencontre championnat[N][N])
{  int i,j;
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
      {  if(i!=j)
         {  championnat[i][j].code_local = i;
            championnat[i][j].code_visiteur = j;
            initialise_score(&(championnat[i][j]));
         }
      }
   }
}
```

Exercice 9

```
void affiche_championnat(t_rencontre championnat[N][N])
{
   int i,j;

   // affiche les noms des equipes
   printf("\t\t");
   for(i=0;i<N;i++)
   {  //affiche_nom(i);
      printf("%-15s",equipes[i].nom);
      printf("\t");
   }
   printf("\n");

   // affiche le contenu de la matrice
   for(i=0;i<N;i++)
   {  // affiche l'equipe traitee
      //affiche_nom(i);
      printf("%-15s",equipes[i].nom);
      printf("\t");

      // affiche les resultats de l'equipe
      for(j=0;j<N;j++)
      {  if(i==j)
            printf("-\t\t");
         else
            printf("%d-
%d\t\t",championnat[i][j].buts_local,championnat[i][j].buts_visiteur);
      }

      printf("\n");
   }
}
```

4 Classement

Exercice 10

```
void raffraichis_points(int points[N], t_rencontre rencontre)
{  // victoire du local
   if(rencontre.buts_local>rencontre.buts_visiteur)
      points[rencontre.code_local] = points[rencontre.code_local] + 3;
   // victoire du visiteur
   else if(rencontre.buts_local<rencontre.buts_visiteur)
      points[rencontre.code_visiteur] = points[rencontre.code_visiteur] + 3;
   // egalite
   else
   {  points[rencontre.code_local] = points[rencontre.code_local] + 1;
      points[rencontre.code_visiteur] = points[rencontre.code_visiteur] + 1;
   }
}
```

Exercice 11

```
void raffraichis_buts(int pour[N], int contre[N], t_rencontre rencontre)
{  // local
   pour[rencontre.code_local] = pour[rencontre.code_local] +
                                                    rencontre.buts_local;
   pour[rencontre.code_visiteur] = pour[rencontre.code_visiteur] +
                                                   rencontre.buts_visiteur;
   // visiteur
   contre[rencontre.code_local] = contre[rencontre.code_local] +
                                                   rencontre.buts_visiteur;
   contre[rencontre.code_visiteur] = contre[rencontre.code_visiteur] +
                                                    rencontre.buts_local;
}
```

Exercice 12

```
void calcule_resultats(int points[N], int pour[N], int contre[N], t_rencontre
                                                          championnat[N][N])
{  int i,j;

   // initialisation des tableaux
   for(i=0;i<N;i++)
   {  pour[i] = 0;
      contre[i] = 0;
      points[i] = 0;
   }

   // calcul des resultats
   for(i=0;i<N;i++)
   {  for(j=0;j<N;j++)
      {  if(i!=j)
         {  raffraichis_points(points,championnat[i][j]);
            raffraichis_buts(pour,contre,championnat[i][j]);
         }
      }
   }
}
```

Exercice 13

```
int compare_equipes(t_code eq1, t_code eq2, int points[N], int pour[N], int
                                                                  contre[N])
{  int dif1, dif2;
   int resultat = points[eq2] - points[eq1];
   if(resultat==0)
   {  dif1 = pour[eq1]-contre[eq1];
      dif2 = pour[eq2]-contre[eq2];
      resultat = dif2-dif1;
      if(resultat==0)
         resultat = pour[eq2]-pour[eq1];
```

```
      }
      return resultat;
   }
```

Exercice 14

```
   void ordonne_equipes(int points[N], int pour[N], int contre[N], t_code
                                                                  classement[N])
   {  int i,j,rang,comp;

      // initialisation de classement
      for(i=0;i<N;i++)
         classement[i] = -1;

      // calcul du classement
      for(i=0;i<N;i++)
      {  rang = 0;
         for(j=0;j<N;j++)
         {  if(i!=j)
            {  comp = compare_equipes(i,j,points,pour,contre);
               if(comp>0)
                  rang++;
            }
         }
         classement[rang] = i;
      }
   }
```

Exercice 15

```
   void  affiche_classement(int  points[N],  int  pour[N],  int  contre[N],  t_code
classement[N])
   {  int i;
      t_code code;
      printf("   %-15s\tPoints\tPour\tContre\tDifference\n","Equipe");
      for(i=0;i<N;i++)
      {  code = classement[i];
         printf("%2d.",i+1);
         printf("%-15s\t",equipes[code].nom);
         printf("%d\t",points[code]);
         printf("%d\t",pour[code]);
         printf("%d\t",contre[code]);
         printf("%3d\n",pour[code]-contre[code]);
      }
   }
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 28 | floutage d'une image

2 Implémentation basique

Exercice 1

Trivial, cf. les TP précédents.

Exercice 2

```
void initialise_matrice_uniforme(float *c, int n)
{  int i,j;
   float valeur = 1.0/(n*n);
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         *(c+n*i+j) = valeur;
   }
}
```

Exercice 3

```
void calcule_voisinage(SDL_Surface* image, int x, int y, Uint32* voisinage, int
n)
{  int i,j,a,b;
   for(i=0;i<n;i++)
   {  a = x+i-n/2;
      for(j=0;j<n;j++)
      {  b = y+j-n/2;
         *(voisinage+n*i+j) = couleur_pixel_surface(image,a,b);
      }
   }
}
```

Exercice 4

```
Uint32 calcule_couleur(SDL_Surface* image, int x, int y, float *c, int n){
   Uint32 voisinage[n][n], resultat;
   int i,j;
   float sr=0, sv=0, sb=0, poids;
   Uint8 r,v,b;

   // on calcule le voisinage de (x,y)
   calcule_voisinage(image, x, y, voisinage, n);

   // on calcule la somme ponderee pour chaque composante
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
      {  convertis_couleur(voisinage[i][j], &r, &v, &b);
         poids = *(c+n*i+j);
         sr = sr + poids*r;
         sv = sv + poids*v;
         sb = sb + poids*b;
      }
   }

   // on calcule la nouvelle couleur
   resultat = convertis_rvb((int)sr, (int)sv, (int)sb);
   return resultat;
}
```

Exercice 5

```
   SDL_Surface* floute_rectangle(SDL_Surface* source, int x, int y, int l, int h,
float *c, int n)
   {  SDL_Surface* cible;
      int i,j;
      Uint32 coul;

      // on clone l'image originale
      cible = clone_surface(source);

      // on traite chaque pixel du rectangle
      for(i=x;i<x+l;i++)
      {  for(j=y;j<y+h;j++)
         {  coul = calcule_couleur(source, i, j, c, n);
            allume_pixel_surface(cible,i,j,coul);
         }
      }

      return cible;
   }
```

3 Gestion des bords

Exercice 6

```
   void calcule_voisinage2(SDL_Surface* image, int x, int y, Uint32* voisinage, int
n)
   {  int i,j,a,b;
      for(i=0;i<n;i++)
      {  a = x+i-n/2;
         if(a<0 || a>=image->w)
            a = x+(n-1-i)-n/2;
         for(j=0;j<n;j++)
         {  b = y+j-n/2;
            if(b<0 || b>=image->h)
               b = y+(n-1-j)-n/2;
            *(voisinage+n*i+j) = couleur_pixel_surface(image,a,b);
         }
      }
   }
```

Exercice 7

```
   SDL_Surface* floute_image(SDL_Surface* source, float *c, int n)
   {  SDL_Surface* resultat = floute_rectangle(source,0,0,source->w,source->h,c,n);
      return resultat;
   }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 29 | flous avancés

1 Flou gaussien

Exercice 1

```
void normalise_matrice(float *c, int n)
{  int i,j;

   // calcul de la somme des valeurs
   float somme = 0;
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         somme = somme + *(c+n*i+j);
   }

   // normalisation des valeurs
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         *(c+n*i+j) = *(c+n*i+j) / somme;
   }
}
```

Exercice 2

```
void initialise_matrice_gaussienne(float *c, int n, float sigma)
{  int i,j;
   float valeur,coef,expo;
   coef = 1 / (pow(sigma,2)*2*PI);

   // on calcule les valeurs d'apres la loi de Gauss
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
      {  expo = - (pow(i-n/2,2) + pow(j-n/2,2)) / (2*pow(sigma,2));
         valeur = coef * pow(EXP,expo);
         *(c+n*i+j) = valeur;
      }
   }

   // on normalise tout ca
   normalise_matrice(c,n);
}
```

2 Flou cinétique

Exercice 4

```
void initialise_matrice_cinetique(float *c, int n, float coef_dir, int
multiplicite)
{  int i,j,k,d;
   float valeur;

   // on met des zeros partout
   for(i=0;i<n;i++)
   {  for(j=0;j<n;j++)
         *(c+n*i+j) = 0;
   }
```

```
    // on trace la droite principale et ses paralleles
    if(coef_dir>-1 && coef_dir<1)
    {  for(k=0;k<multiplicite;k++)
       {  valeur = multiplicite - k;
          for(d=-1;d<=1;d=d+2)
          {  for(i=0;i<n;i++)
             {  j = (int)(coef_dir*(i-n/2)) + n/2 + d*k;
                if(j>=0 && j<n)
                   *(c+n*i+j) = valeur;
             }
          }
       }
    }
    else
    {  for(k=0;k<multiplicite;k++)
       {  valeur = multiplicite - k;
          for(d=-1;d<=1;d=d+2)
          {  for(j=0;j<n;j++)
             {  i = (int)((j-n/2)/coef_dir) + n/2 + d*k;
                if(i>=0 && i<n)
                   *(c+n*i+j) = valeur;
             }
          }
       }
    }

    // on normalise
    normalise_matrice(c,n);
}
```

3 Flou radial

Exercice 6

```
float calcule_coefficient_directeur(int x1, int y1, int x2, int y2)
{  float resultat;
   float numerateur = y2 - y1;
   float denominateur = x2 - x1;

   if(denominateur==0)
   {  if(numerateur==0)
         resultat = 0;
      else
         resultat = FLT_MAX;
   }
   else
      resultat = numerateur / denominateur;

   return resultat;
}
```

Exercice 7

```
SDL_Surface* floute_image_radial(SDL_Surface* source, int x, int y)
{  SDL_Surface* cible;
   int i,j,n=11,multiplicite=1;
   float c[n][n],coef_dir;
   Uint32 coul;

   // on clone l'image originale
   cible = clone_surface(source);

   // on traite chaque pixel du rectangle
   for(i=0;i<source->w;i++)
   {  for(j=0;j<source->h;j++)
      {  coef_dir = calcule_coefficient_directeur(x,y,i,j);
         initialise_matrice_cinetique(c,n,coef_dir,multiplicite);
         coul = calcule_couleur(source, i, j, c, n);
```

```
                allume_pixel_surface(cible,i,j,coul);
            }
        }

        return cible;
    }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

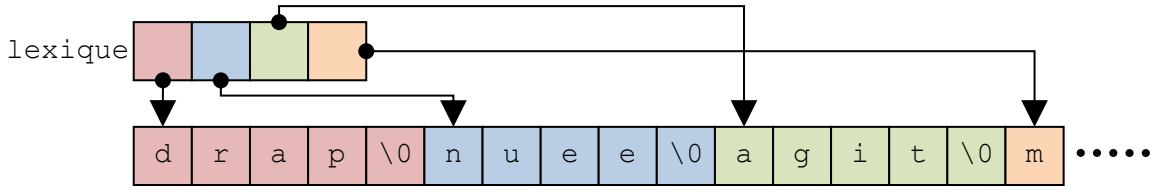
Damien Berthet & Vincent Labatut

Correction TP 30 | gestion d'un lexique

1 Définition du lexique

Exercice 1

La variable `lexique` désigne un tableau de pointeurs sur des chaînes de caractères constantes.



- `*lexique` est le contenu de la première case du tableau `lexique`, i.e. un pointeur sur une chaîne de caractères, de type `char*`. Une chaine de caractères est un tableau, donc il s'agit en fait de l'adresse de la première case de ce tableau. Autrement dit, cette expression est l'adresse de la première lettre du lexique (ici, la lettre `'d'`).
- `lexique+1` est l'adresse de la deuxième case du tableau `lexique`. C'est donc l'adresse d'un pointeur sur la chaine `"nuee"`, et son type est par conséquent `char**`.
- `*(lexique+2)` représente le contenu de la 3ème case du tableau `lexique`, et son type est donc `char*`. Comme pour `*lexique`, on obtient l'adresse de la première lettre de la chaîne pointée (i.e. `'a'`, dans notre cas).
- `*(lexique+1)+1` est l'adresse de la 2ème lettre de la chaîne pointée par le pointeur `*(lexique+1)`. Son type est donc `char*`, et il s'agit de l'adresse de la lettre `'u'` dans le mot `"nuee"`.
- `*(*(lexique+2)+2)` est une valeur littérale de type `char` et sa valeur est `'i'` : 3ème lettre du 3ème mot de `lexique`.
- De la même façon, `*(*(lexique+3)+4)` est la 5ème lettre du 3ème mot, donc une valeur de type `char` correspondant au caractère de fin de chaîne `'\0'`.

Exercice 2

```
void affiche_lexique()
{  int i;
   for(i=0;i<TAILLE_LEXIQUE;i++)
      printf("lexique[%d]=\"%s\"\n",i,lexique[i]);
}
```

2 Recherche dans le lexique

Exercice 3

```
int cherche_prefixe(char *prefixe)
{  int i,resultat=0;
   printf("La chaine \"%s\" est un prefixe des mots suivants :\n",prefixe);
```

```
    for(i=0;i<TAILLE_LEXIQUE;i++)
    {  if(est_prefixe(prefixe,lexique[i]))
       {  resultat++;
          printf("%3d.lexique[%d]=\"%s\"\n",resultat,i,lexique[i]);
       }
    }

    return resultat;
}
```

Exercice 4

```
int cherche_premier()
{  int i,comp,resultat=1;

    for(i=1;i<TAILLE_LEXIQUE;i++)
    {  comp = compare_chaines(lexique[resultat],lexique[i]);
       if(comp>0)
          resultat = i;
    }

    return resultat;
}
```

Exercice 5

```
int cherche_premier_partiel(int debut)
{  int i,comp,resultat=debut;

    for(i=debut;i<TAILLE_LEXIQUE;i++)
    {  comp = compare_chaines(lexique[resultat],lexique[i]);
       if(comp>0)
          resultat = i;
    }

    return resultat;
}
```

3 Tri du lexique

Exercice 6

```
void permute_mots(int i, int j)
{  char *p = lexique[i];
   lexique[i] = lexique[j];
   lexique[j] = p;
}
```

Exercice 7

```
void trie_lexique()
{  int i,premier;

   for(i=0;i<TAILLE_LEXIQUE;i++)
   {  premier = cherche_premier_partiel(i);
      permute_mots(i,premier);
   }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 31 | allocation dynamique

1 Organisation de la mémoire

Exercice 1

Le programme modifié est (les ajouts sont indiqué en turquoise) :

```
void fonction()
{  short g,h,i;
   short *p,*q,*r;
   if((p=(short*)malloc(sizeof(short)))==NULL
      || (q=(short*)malloc(sizeof(short)))==NULL
      || (r=(short*)malloc(sizeof(short)))==NULL)
      printf("fonction:malloc: ERREUR lors de l'allocation.\n");
   printf("&g:%p &h:%p &i:%p \n",&g,&h,&i);
   printf(" p:%p  q:%p  r:%p \n",p,q,r);
}

int main()
{  short d,e,f;
   short *m,*n,*o,*s,*t,*u;
   if((m=(short*)malloc(sizeof(short)))==NULL
      || (n=(short*)malloc(sizeof(short)))==NULL
      || (o=(short*)malloc(sizeof(short)))==NULL)
      printf("main:malloc: ERREUR lors de l'allocation.\n");
   printf("&a:%p &b:%p &c:%p \n",&a,&b,&c);
   printf("&d:%p &e:%p &f:%p \n",&d,&e,&f);
   printf(" m:%p  n:%p  o:%p \n",m,n,o);

   fonction();

   short j,k,l;
   if((s=(short*)malloc(sizeof(short)))==NULL
      || (t=(short*)malloc(sizeof(short)))==NULL
      || (u=(short*)malloc(sizeof(short)))==NULL)
      printf("main:malloc: ERREUR lors de l'allocation.\n");
   printf("&j:%p &k:%p &l:%p \n",&j,&k,&l);
   printf(" s:%p  t:%p  u:%p \n",s,t,u);

    return EXIT_SUCCESS;
}
```

Vous remarquerez que pour les pointeurs, on affiche directement leur contenu (qui est une adresse) et non pas leur adresse.

Avant la modification, on a un résultat du type :

```
&a:0x601030        &b:0x601032        &c:0x601034
&d:0x7fff23cbc8ce  &e:0x7fff23cbc8cc  &f:0x7fff23cbc8ca
&g:0x7fff23cbc89e  &h:0x7fff23cbc89c  &i:0x7fff23cbc89a
&j:0x7fff23cbc8c8  &k:0x7fff23cbc8c6  &l:0x7fff23cbc8c4
```

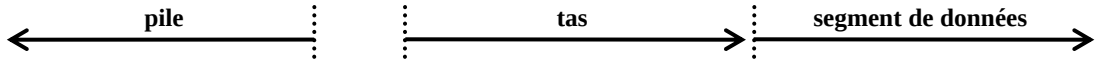
Après la modification, on a :

```
&a:0x601040        &b:0x601042        &c:0x601044
&d:0x7fffd7b2a68e  &e:0x7fffd7b2a68c  &f:0x7fffd7b2a68a
 m:0x1b72010        n:0x1b72030        o:0x1b72050
&g:0x7fffd7b2a62e  &h:0x7fffd7b2a62c  &i:0x7fffd7b2a62a
 p:0x1b72070        q:0x1b72090        r:0x1b720b0
&j:0x7fffd7b2a688  &k:0x7fffd7b2a686  &l:0x7fffd7b2a684
 s:0x1b720d0        t:0x1b720f0        u:0x1b72110
```

On peut faire les remarques suivantes :

- On peut observer la conséquence sur les adresses de `j`, `k` et `l` de la déclaration des pointeurs dans le `main` : les adresses sont décalées par rapport à avant la modification.
- Le début du tas (`0x1b72010`) est localisé entre le début de la pile (`0x7fffd7b2a68e`) et celui du segment de données (`0x601040`).
- Comme pour le segment de données, les adresses des nouvelles zones allouées sont obtenues par incrémentation (`0x1b72010`, `0x1b72030`, `0x1b72050`, `0x1b72070`…), alors que c'est par décrémentation pour la pile (`0x7fffd7b2a68e`, `0x7fffd7b2a68c`, `0x7fffd7b2a68a`…).

La représentation graphique de la mémoire ressemble donc à :



Attention : l'organisation exacte dépend du contexte (compilateur, système d'exploitation, etc.). En particulier, le tas n'est pas forcément localisé entre la pile et le segment de données.

2 Tableaux d'entiers

Exercice 2

- Fonction `affiche_tableau` :

```
void affiche_tableau(short *tab, int taille)
{  int i;
   printf("{");
   for(i=0;i<taille;i++)
      printf(" %d",tab[i]);
   printf(" }");
}
```

- Fonction `alloue_tableau1` :

```
short* alloue_tableau1(int taille)
{  short *p,i;
   // réservation de l'espace mémoire
   if((p = (short*)malloc(taille*sizeof(short))) == NULL)
      printf("alloue_tableau1: erreur lors du malloc\n");
   else
   {  // initialisation à 1
      for(i=0;i<=taille;i++)
         p[i]=1;
   }
   return p;
}
```

- Fonction `main` :

```
int main(int argc, char **argv)
{  short *t;
   t = alloue_tableau1(N);
   affiche_tableau(t,N);printf("\n");

   exit(EXIT_SUCCESS);
}
```

Exercice 3

- Fonction `alloue_tableau2` :

```
void alloue_tableau2(int taille, short **tab)
{  short *p,i;
   // réservation de l'espace mémoire
   if((p = (short*)malloc(taille*sizeof(short))) == NULL)
      printf("alloue_tableau2 : erreur lors du malloc\n");
```

```
      else
      {  // initialisation à 0
         for(i=0;i<=taille;i++)
         p[i]=1;
      }
      *tab = p;
   }
```

- Fonction `main` :

```
   int main(int argc, char **argv)
   {  short *t;
      alloue_tableau2(N,&t);
      affiche_tableau(t,N);printf("\n");

      exit(EXIT_SUCCESS);
   }
```

3 Chaînes de caractères

Exercice 4

- Fonction `saisis_chaine_tampon` :

```
   void saisis_chaine_tampon(char **chaine)
   {  char c[100],*p;
      int i=0,taille;
      // saisie de la chaine
      gets(c);
      // calcul de la taille de la chaine
      while(c[i]!='\0')
         i++;
      taille=i+1;
      // réservation de l'espace mémoire
      if((p = (char*)malloc(taille*sizeof(char))) == NULL)
         printf("saisis_chaine_tampon: erreur lors du malloc\n");
      else
      {  // copie de la chaine
         for(i=0;i<=taille;i++)
            p[i]=c[i];
         // mise à jour du paramètre
         *chaine = p;
      }
   }
```

- Fonction `main` :

```
   int main(int argc, char **argv)
   {  char* chaine;
      printf("Entrez une chaine de caracteres :\n");
      saisis_chaine_tampon(&chaine);
      printf("La chaine saisie est : \"%s\"\n",chaine);

      exit(EXIT_SUCCESS);
   }
```

Exercice 5

- Fonction `saisis_chaine_direct` :

```
   void saisis_chaine_direct(char **chaine)
   {  char c,*p=NULL;
      int i=0;
      do
      {  c=getche();
         if((p = (char*)realloc(p, (i+1)*sizeof(char))) == NULL)
            printf("saisis_chaine_direct: erreur lors du realloc\n");
         else
         {  p[i]=c;
            i++;
         }
```

```
    }
    while(c!='\r' /*pour windows*/ && c!='\n' /*pour linux*/);
    p[i-1]='\0';
    *chaine = p;
}
```

- Fonction `main` :

```
int main(int argc, char **argv)
{  char* chaine;
   printf("Entrez une chaine de caracteres :\n");
   saisis_chaine_direct(&chaine);
   printf("La chaine saisie est : \"%s\"\n",chaine);

   exit(EXIT_SUCCESS);
}
```

4 Tableaux de pointeurs

Exercice 6

```
void remplis_mots()
{  int i;

   // saisie
   for(i=0;i<N;i++)
   {  printf("Entrez le mot n.%d : ",i);
      saisis_chaine_tampon(&mots[i]);
   }

   // affichage
   for(i=0;i<N;i++)
      printf("Mot n.%d : %s\n",i,mots[i]);
}
```

Correction TP 32 | générateur pseudo-aléatoire

2 Implémentation

Exercice 1

- Fichier `alea.h` :

```
#ifndef ALEA_H_
#define ALEA_H_

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

#define A 137
#define C 187
#define M 256

unsigned char genere_nombre_lehmer();

#endif /* ALEA_H_ */
```

- Fichier `alea.c` :

```
#include "alea.h"

unsigned char u = 0;

unsigned char genere_nombre_lehmer()
{  u = (u*A + C) % M;
   return u;
}
```

Remarquez que `A`, `C` et `M` sont des constantes définies dans `alea.h`, alors que `u` est une variable globale définie dans `alea.c`.

Exercice 2

```
unsigned char init_graine_lehmer()
{  time_t t;
   if(time(&t)==-1)
      printf("init_graine_lehmer: erreur lors de l'execution de time()\n");
   else
      u = t%M;
   return u;
}
```

Exercice 3

```
unsigned char* genere_tableau_lehmer(int n)
{  int i;
   unsigned char *resultat;
   if((resultat=(unsigned char*)malloc(n*sizeof(unsigned char))) == NULL)
      printf("genere_tableau_lehmer: erreur lors du malloc\n");
   else
   {  for(i=0;i<=n;i++)
         resultat[i] = genere_nombre_lehmer();
   }
   return resultat;
```

```
}
```

3 Vérification

Exercice 4

```
void calcule_stats(unsigned char *tab, int n, int *min, int *max, float *moy,
float *et)
{  int i;
   float somme = tab[0];
   *min = tab[0];
   *max = tab[0];

   for(i=1;i<n;i++)
   {  somme = somme + tab[i];
      if(tab[i]<*min)
         *min = tab[i];
      if(tab[i]>*max)
         *max = tab[i];
   }
   *moy = somme/n;

   somme = 0;
   for(i=0;i<n;i++)
      somme = somme + pow(tab[i]-*moy,2);
   *et = sqrt(somme/(n-1));
}
```

Exercice 5

```
void calcule_repartition(unsigned char *tab, int n, int **dist)
{  int i;
   if((*dist=(int*)malloc(M*sizeof(int))) == NULL)
      printf("calcule_repartition: erreur lors du malloc\n");
   else
   {  for(i=0;i<M;i++)
         (*dist)[i] = 0;
      for(i=0;i<=n;i++)
         (*dist)[tab[i]]++;
   }
}
```

Exercice 6

Fichier `main.c` demandé :

```
#include <stdio.h>
#include <stdlib.h>

#include "alea.h"
#include "histogramme.h"

int main(int argc, char *argv[])
{  int g,n,*dist;
   unsigned char *tab;
   g = init_graine_lehmer();
   printf("graine: %d\n",g);
   n = 10000;
   tab = genere_tableau_lehmer(n);
   calcule_repartition(tab,n,&dist);
   histogramme_horizontal(dist,M);

   return EXIT_SUCCESS;
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 33 | nombres hexadécimaux

2 Opérations simples

Exercice 1

```
int longueur_nombre(char *n)
{   int i=0;
    while(n[i]!='\0')
       i++;
    return i;
}
```

Exercice 2

```
int verifie_nombre(char *n)
{  int resultat=1;
   int i=0;
   while(n[i]!='\0' && resultat)
   {  if((n[i]>='0' && n[i]<='9') || (n[i]>='A' && n[i]<='F'))
         i++;
      else
         resultat=0;
   }
   return resultat;
}
```

3 Comparaisons

Exercice 3

```
int compare_chiffres(char c1, char c2)
{  int resultat;
   if(c1>='0' && c1<='9')
      if(c2>='0' && c2<='9')
         // c1 et c2 sont des chiffres
         resultat=c1-c2;
      else
         // c1 est un chiffre, c2 est une lettre
         resultat=-1;
   else
      if(c2>='0' && c2<='9')
         // c1 est une lettre, c2 est un chiffre
         resultat=1;
      else
         // c1 et c2 sont des lettres
         resultat=c1-c2;
   return resultat;
}
```

Exercice 4

```
int compare_nombres(char *n1, char *n2)
{  int l1=longueur_nombre(n1),l2=longueur_nombre(n2);
   int resultat=l1-l2,i=0;
   if(resultat==0)
   {  while(i<l1 && n1[i]==n2[i])
         i++;
```

```
        if(i==l1)
           resultat=0;
        else
           resultat=compare_chiffres(n1[i],n2[i]);
     }
     return resultat;
  }
```

4 Conversion

Exercice 5

```
  char convertit_chiffre(int x)
  {  char resultat;
     if(x<10)
        resultat='0'+x;
     else
        resultat='A'+(x-10);
     return resultat;
  }
```

Exercice 6

```
  char* convertit_nombre(int x)
  {  int taille=0,temp=x,reste,i=0;
     char *resultat,chiffre;
     // on calcule le nombre de chiffres du résultat
     while(temp!=0)
     {  temp=temp/16;
        taille++;
     }
     // on rajoute l'espace pour le '\0'
     taille++;
     // allocation dynamique
     if((resultat = (char *) malloc(sizeof(char)*taille))==NULL)
        printf("convertit_nombre : erreur lors du malloc\n");
     // construction du nombre en base 16
     temp=x;
     resultat[taille-1]='\0';
     while(temp!=0)
     {  // extraction du dernier chiffre
        reste=temp%16;
        temp=temp/16;
        // conversion du chiffre en base 16
        chiffre = convertit_chiffre(reste);
        // on place le chiffre dans le nombre
        resultat[taille-2-i]=chiffre;
        i++;
     }
     return resultat;
  }
```

5 Modifications

Exercice 7

```
  void incremente_nombre(char **n)
  {  int i,longueur;
     char *p=*n,retenue='0',*q;
     // on calcule la longueur du nombre
     longueur=longueur_nombre(p);
     i=longueur-1;
     // on effectue l'incrémentation
     do
     {  switch(p[i])
        {  case 'F':
              retenue='1';
```

```
                p[i]='0';
                break;
            case '9':
                retenue='0';
                p[i]='A';
                break;
            default:
                retenue='0';
                p[i]++;
        }
        i--;
    }
    while(i>=0 && retenue=='1');
    // s'il y a une retenue, on agrandit la mémoire allouée à la chaîne
    if(retenue!='0')
    {  if((q = (char *)realloc(p,sizeof(char)*longueur+1))==NULL)
          printf("incremente : erreur lors du realloc\n");
       else
       {  for(i=longueur-1;i>0;i--)
             q[i]=p[i-1];
          q[i]=retenue;
          *n=q;
       }
    }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 34 | répertoire téléphonique

1 Numéros de téléphone

Exercice 2

```
void affiche_numero(int* numero)
{  int *p = numero;
   while(*p!=FIN)
   {  printf("%d",*p);
      p++;
   }
}
```

Exercice 3

```
void saisis_numero(int** numero)
{  char c;
   int *p=NULL;
   int i = 0;
   do
   {  // on saisit un seul caractere
      c = getche();
      // on verifie si c'est un chiffre
      if(c>='0' && c<='9')
      {  // on agrandit le tableau courant
         if((p = (int*)realloc(p, (i+1)*sizeof(int))) == NULL)
            printf("saisis_numero: erreur lors du realloc\n");
         // on y place le nouveau chiffre
         else
         {  p[i] = c - '0';
            i++;
         }
      }
   }
   while(c!='\r' /*pour windows*/ && c!='\n' /*pour linux*/);
   // on rajoute le -1 final
   p[i] = FIN;
   // on passe le resultat pas adresse
   *numero = p;
}
```

2 Contacts

Exercice 5

- Fonction d'affichage :

```
void affiche_contact(t_contact contact)
{  printf("%s : ",contact.nom);
   affiche_numero(contact.numero);
}
```

- Fonction de saisie :

```
void saisis_contact(t_contact* contact)
{  // nom
   char c,*p=NULL;
   int i = 0;
   printf("Entrez le nom du contact : ");
```

```
    do
    {  c = getche();
       if((p = (char*)realloc(p, (i+1)*sizeof(char))) == NULL)
          printf("saisis_chaine_direct: erreur lors du realloc\n");
       else
       {  p[i]=c;
          i++;
       }
    }
    while(c!='\r' /*pour windows*/ && c!='\n' /*pour linux*/);
    p[i-1]='\0';
    contact->nom = p;

    // numero
    printf("Entrez le numero de %s : ",contact->nom);
    saisis_numero(&(contact->numero));
}
```

3 Répertoire

Exercice 7

```
void initialise_repertoire (t_repertoire* repertoire)
{  repertoire->contacts = NULL;
   repertoire->taille = 0;
}
```

Exercice 8

```
void ajoute_contact(t_repertoire* repertoire, t_contact contact)
{  int taille = repertoire->taille;
   t_contact *p = repertoire->contacts;

   if((p = (t_contact*)realloc(p, (taille+1)*sizeof(t_contact))) == NULL)
      printf("ajoute_contact: erreur lors du realloc\n");
   else
   {  // ici on peut faire une affectation superficielle entre deux valeurs
      // de type structure (ce qui n'est pas toujours vrai, en general)
      p[taille] = contact;
      repertoire->taille = taille + 1;
      repertoire->contacts = p;
   }
}
```

Exercice 9

```
void affiche_repertoire(t_repertoire repertoire)
{  int i;

   printf("Taille du repertoire : %d contact",repertoire.taille);
   if(repertoire.taille>1)
      printf("s");
   printf(".\n");

   for(i=0;i<repertoire.taille;i++)
   {  printf("%3d. ",i+1);
      affiche_contact(repertoire.contacts[i]);
      printf("\n");
   }
}
```

Exercice 10

```
t_contact* cherche_nom(t_repertoire repertoire, char* nom)
{  int i=0, comp;
   t_contact *resultat = NULL;
   while(resultat==NULL && i<repertoire.taille)
   {  comp = compare_chaines(nom,repertoire.contacts[i].nom);
      if(comp==0)
```

```
            resultat = &repertoire.contacts[i];
         else
            i++;
      }
      return resultat;
   }
```

Exercice 11

```
   void supprime_contact(t_repertoire* repertoire, t_contact* contact)
   {  //  on  calcule  la  position  du  contact  en  utilisant  l'arithmetique  des
pointeurs
      // on suppose que le parametre contact n'est pas NULL et que ce contact
      // apartient bien au repertoire.
      int index = contact-repertoire->contacts;

      int taille = repertoire->taille;
      int i;
      for(i=index;i<taille-1;i++)
         repertoire->contacts[i] = repertoire->contacts[i+1];

      repertoire->taille = taille - 1;
   }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 35 | fichiers et arguments d'un programme

1 Arguments d'un programme

Exercice 1

```
void affiche_arguments(int argc, char *argv[])
{  int i;
   printf("Nombre d'arguments recus : %d\n",argc);
   printf("Nom du programme : %s\n",argv[0]);
   for(i=1;i<argc;i++)
      printf("Argument %d : %s\n",i,argv[i]);
}
```

2 Accès non-formaté à un fichier

Exercice 3

```
int ecris_chaine()
{  int resultat=0;

   // on saisit la chaine
   char chaine[10];
   printf("entrez la chaine : ");
   scanf("%s",chaine);

   // on ouvre le fichier
   FILE *fp;
   if((fp = fopen("donnees.txt", "w")) == NULL)
   {  //traitement en cas d'erreur
      printf("ecris_chaine: erreur dans fopen\n");
      return -1;
   }

   // on écrit la chaîne
   while(chaine[resultat] !='\0')
   {  if((fputc(chaine[resultat],fp)) == EOF)
      {  // traitement de l'erreur
         printf("ecris_chaine: erreur dans fputc\n");
         return -1;
      }
      resultat++;
   }

   // on ferme le fichier
   if((fclose(fp)) == EOF)
   {  // traitement de l'erreur
      printf("ecris_chaine: erreur dans fclose\n");
      return -1;
   }

   return resultat;
}
```

Exercice 4

```
int lis_chaine()
{  char chaine[10];
```

```
    FILE *fp;
    int resultat = 0;

    // on ouvre le fichier
    if((fp = fopen("donnees.txt", "r")) == NULL)
    {  //traitement en cas d'erreur
       printf("lis_chaine: erreur dans fopen\n");
       return -1;
    }

    // on lit la chaîne
    while((chaine[resultat]=fgetc(fp)) != EOF)
       resultat++;
    if(!feof(fp))
    {  // traitement de l'erreur
       printf("lis_chaine: erreur dans fgetc\n");
       return -1;
    }
    chaine[resultat]='\0';

    // on ferme le fichier
    if((fclose(fp)) == EOF)
    {  // traitement de l'erreur
       printf("lis_chaine: erreur dans fclose\n");
       return -1;
    }

    // on affiche la chaine
    printf("La chaine lue est : \"%s\"\n",chaine);

    return resultat;
}
```

Exercice 5

```
int copie_fichier(char *source, char *cible)
{  int resultat = 0;

   // on ouvre le fichier source en lecture
   FILE *src;
   if((src = fopen(source, "r")) == NULL)
   {  //traitement en cas d'erreur
      printf("copie_fichier: erreur dans fopen\n");
      return -1;
   }
   // on ouvre le fichier cible en écriture
   FILE *cbl;
   if((cbl = fopen(cible, "w")) == NULL)
   {  //traitement en cas d'erreur
      printf("copie_fichier: erreur dans fopen\n");
      return -1;
   }

   // on lit la source ligne par ligne
   // en écrivant chaque ligne dans la cible
   char tampon[50];
   while(fgets(tampon,50,src) != NULL)
   {  if((fputs(tampon,cbl)) == EOF)
      {  // traitement de l'erreur
         printf("copie_fichier: erreur dans fputs\n");
         return -1;
      }
      printf("%s",tampon);
      resultat++;
   }
   if(!feof(src))
   {  // traitement de l'erreur
      printf("copie_fichier: erreur dans fgets\n");
```

```
        return -1;
    }

    // on ferme les fichiers
    if((fclose(src)) == EOF)
    {  // traitement de l'erreur
       printf("copie_fichier: erreur dans fclose\n");
       return -1;
    }
    if((fclose(cbl)) == EOF)
    {  // traitement de l'erreur
       printf("copie_fichier: erreur dans fclose\n");
       return -1;
    }

    printf("\n");
    return resultat;
}
```

Exercice 6

```
int main(int argc, char *argv[])
{  if(argc<3)
   {  printf("Erreur: Il n'y a pas assez de paramètres (%d au lieu de 2)
                                                                \n",argc-1);
      exit(EXIT_FAILURE);
   }
   else if(argc>3)
   {  printf("Erreur: Il y a trop de paramètres (%d au lieu de 2)\n",argc-1);
      exit(EXIT_FAILURE);
   }
   else
      copie_fichier(argv[1],argv[2]);

   return EXIT_SUCCESS;
}
```

3 Accès formaté à un fichier

Exercice 7

```
int sauve_date(t_date d, FILE *fp)
{  int resultat = 0;
   if(fprintf(fp,"%d/%d/%d",d.jour,d.mois,d.annee) == EOF)
   {  // traitement de l'erreur
      printf("sauve_date: erreur dans fprintf\n");
      resultat = -1;
   }
   return resultat;
}
```

Exercice 8

```
int charge_date(t_date *d, FILE *fp)
{  int resultat = 0;
   if(fscanf(fp,"%d/%d/%d",&(d->jour),&(d->mois),&(d->annee))== EOF)
   {  // traitement de l'erreur
      printf("charge_date: erreur dans fscanf\n");
      resultat = -1;
   }
   return resultat;
}
```

Exercice 9

La modification réalisée dans le fichier introduit une valeur trop grande pour le type entier, ce qui explique la valeur incorrecte (par rapport à celle du fichier) affichée.

Exercice 10

Lors de la lecture, `fscanf` tente absolument de respecter le format, et cherche donc des `'/'` qui n'existent pas, ce qui provoque l'affichage de valeurs erronées.

Exercice 11

- Enregistrement d'un étudiant dans un fichier :

```
int sauve_etudiant(t_etudiant e, FILE *fp)
{  int resultat = 0;
   if(fprintf(fp,"%s\t%s\t%d\t",e.prenom,e.nom,e.genre) == EOF)
   {  printf("sauve_etudiant: erreur dans fprintf\n");
      resultat = -1;
   }
   else
      resultat = sauve_date(e.naissance,fp);
   return resultat;
}
```

- Chargement d'un étudiant depuis un fichier :

```
int charge_etudiant(t_etudiant *e, FILE *fp)
{  int resultat = 0;
   if(fscanf(fp,"%s\t%s\t%d\t",e->prenom,e->nom,&(e->genre))== EOF)
   {  printf("charge_etudiant: erreur dans fscanf\n");
      resultat = -1;
   }
   else
      resultat = charge_date(&(e->naissance),fp);
   return resultat;
}
```

Exercice 12

- Enregistrement d'une promotion dans un fichier :

```
int sauve_promotion(t_promo p, FILE *fp)
{  int resultat=0,temp;

   // on sauve la taille de la promotion
   if(fprintf(fp,"%d\n",p.taille) == EOF)
   {  printf("sauve_promotion: erreur dans fprintf\n");
      resultat = -1;
   }

   // puis on sauve chaque etudiant
   while(resultat>-1 && resultat<p.taille)
   {  temp = sauve_etudiant(p.etudiants[resultat],fp);
      if(temp==-1)
         resultat = temp;
      else
      {  if(fprintf(fp,"\n") == EOF)
         {  printf("sauve_promotion: erreur dans fprintf\n");
            resultat = -1;
         }
         else
            resultat++;
      }
   }

   return resultat;
}
```

- Chargement d'une promotion depuis un fichier :

```
int charge_promotion(t_promo *p, FILE *fp)
{  int resultat=0,temp;

   // on charge la taille de la promotion
   if(fscanf(fp,"%d\n",&(p->taille))== EOF)
   {  printf("charge_promotion: erreur dans fscanf\n");
```

```
            resultat = -1;
        }

        // puis on charge chaque etudiant
        while(resultat>-1 && resultat<p->taille)
        {   temp = charge_etudiant(&(p->etudiants[resultat]),fp);
            if(temp==-1)
                resultat = temp;
            else
            {   if(fscanf(fp,"\n") == EOF)
                {   printf("sauve_promotion: erreur dans fscanf\n");
                    resultat = -1;
                }
                else
                    resultat++;
            }
        }

        return resultat;
    }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 36 | diaporama

1 Noms des images

Exercice 1

- Fonction `lis_nom` :

```
int lis_nom(FILE *fp, char** nom)
{  int taille = 1, resultat = 1;
   char c;

   // on alloue l'espace pour le \0
   if((*nom = (char*)malloc(taille*sizeof(short))) == NULL)
   {  printf("lis_nom: erreur lors du malloc\n");
      exit(EXIT_FAILURE);
   }

   // lecture de chaque caractere un par un
   // on s'arrete en cas d'erreur, de fin de fichier, ou de fin de ligne
   while((c=fgetc(fp))!=EOF && c!='\n')
   {  // augmentation de la taille du tableau contenant la chaine
      taille++;
      if((*nom = (char*)realloc(*nom, (taille)*sizeof(char))) == NULL)
      {  printf("lis_nom: erreur lors du realloc\n");
         exit(EXIT_FAILURE);
      }
      // copie du nouveau caractere
      (*nom)[taille-2] = c;
   }

   // traitement de la fin de fichier
   if(c==EOF)
   {  if(feof(fp))
         resultat = 0;
      else
      {  printf("lis_nom: erreur dans fgetc\n");
         exit(EXIT_FAILURE);
      }
   }

   // ajout du caractere de fin
   (*nom)[taille-1] ='\0';

   return resultat;
}
```

- Fonction `main` :

```
int main(int argc, char** argv)
{  FILE *fp = fopen("album/images.txt", "r");
   char* nom;
   int res = lis_nom(fp, &nom);
   printf("res=%d - nom=%s\n",res,nom);
   res = lis_nom(fp, &nom);
   printf("res=%d - nom=%s\n",res,nom);
   res = lis_nom(fp, &nom);
   printf("res=%d - nom=%s\n",res,nom);
   res = lis_nom(fp, &nom);
   printf("res=%d - nom=%s\n",res,nom);
```

```
    return EXIT_SUCCESS;
}
```

Exercice 2

- Fonction `charge_liste` :

```
int charge_liste(char fichier[], char*** images)
{  FILE *fp;
   int resultat = 0;
   char *nom;

   // on ouvre le fichier
   if((fp = fopen(fichier, "r")) == NULL)
   {  //traitement en cas d'erreur
      printf("charge_liste: erreur dans fopen\n");
      return -1;
   }

   // on lit chaque chaîne
   while(lis_nom(fp,&nom))
   {  if((*images = (char**)realloc(*images, (resultat+1)*sizeof(char*))) ==
                                                                        NULL)
      {  printf("charge_liste: erreur lors du realloc\n");
         exit(EXIT_FAILURE);
      }
      (*images)[resultat] = nom;
      resultat++;
   }

   // on ferme le fichier
   if((fclose(fp)) == EOF)
   {  // traitement de l'erreur
      printf("lis_chaine: erreur dans fclose\n");
      return -1;
   }

   return resultat;
}
```

- Fonction `main` :

```
int main(int argc, char** argv)
{  char** images=NULL;
   int i,res;
   res = charge_liste("album/images.txt",&images);
   for(i=0;i<res;i++)
      printf("%d. %s\n",i,images[i]);

   return EXIT_SUCCESS;
}
```

2 Chemins des images

Exercice 3

- Fonction `ajoute_dossier` :

```
void ajoute_dossier(char* images[], int n, char dossier[])
{  int longueur=0,lg,i,j;

   // on calcule la longueur du prefixe
   longueur = mesure_chaine(dossier) + 1;

   // on traite chaque nom contenu dans le tableau
   for(i=0;i<n;i++)
   {  // on rallonge la chaine
      lg = mesure_chaine(images[i]) + 1;
      if((images[i] = (char*)realloc(images[i], (lg+longueur)*sizeof(char))) ==
```

```
                                                                                NULL)
        {  printf("ajoute_dossier: erreur lors du realloc\n");
           exit(EXIT_FAILURE);
        }
        // on deplace les caracteres du nom
        for(j=lg;j>=0;j--)
           images[i][j+longueur] = images[i][j];
        // on copie le prefixe au debut
        for(j=0;j<longueur-1;j++)
           images[i][j] = dossier[j];
        // on n'oublie pas le separateur entre dossier et fichier
        images[i][j] = '/';
     }
  }
```

- Fonction `main` :

```
int main(int argc, char** argv)
{  char** images=NULL;
   int i,res;
   res = charge_liste("album/images.txt",&images);
   ajoute_dossier(images, res, "album");
   for(i=0;i<res;i++)
      printf("%d. %s\n",i,images[i]);

   return EXIT_SUCCESS;
}
```

3 Affichage des images

Exercice 4

- Fonction `diaporama_image` :

```
void diaporama_image(char *chemin, int delai)
{  int x,y;

   // chargement de l'image et calcule des coordonnees
   SDL_Surface* surface = charge_image(chemin);
   x = (FENETRE_LARGEUR - surface->w) / 2;
   y = (FENETRE_HAUTEUR - surface->h) / 2;

   // on efface la fenetre avant de dessiner l'image
   efface_fenetre();
   dessine_surface(x,y,surface);

   // on attend le delai specifie
   attends_delai(delai);
}
```

- Fonction `main` :

```
int main(int argc, char** argv)
{  initialise_fenetre("Diaporama");
   diaporama_image("album/image1.bmp", 4000);

   return EXIT_SUCCESS;
}
```

Exercice 5

- Fonction `diaporama_tout` :

```
void diaporama_tout(char fichier[], char dossier[], int delai)
{  char *liste,**images=NULL;
   int n,i,l1,l2;

   // construction de la chaine
   l1 = mesure_chaine(dossier);
   l2 = mesure_chaine(fichier);
   if((liste = (char*)malloc(l1+1+l2+1*sizeof(char))) == NULL)
```

```
      {  printf("diaporama_tout: erreur lors du malloc\n");
         exit(EXIT_FAILURE);
      }
      for(i=0;i<l1;i++)
         liste[i] = dossier[i];
      liste[i] = '/';
      for(i=0;i<l2;i++)
         liste[l1+1+i] = fichier[i];
      liste[l1+1+i] = '\0';

      // chargement de la liste
      n = charge_liste(liste,&images);
      // ajout du dossier au debut des chemins
      ajoute_dossier(images, n, dossier);

      // chargement et affichage des images
      for(i=0;i<n;i++)
         diaporama_image(images[i],delai);
   }
```

- Fonction `main` :

```
   int main(int argc, char** argv)
   {  initialise_fenetre("Diaporama");
      diaporama_tout("images2.txt", "album", 1000);
      attends_touche();

      return EXIT_SUCCESS;
   }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 37 | stock d'une librairie

1 Structure de données

Exercice 1

```
#define LONG_CODE 6
#define N 10

typedef struct
{  char code[LONG_CODE];
   t_date publication;
   int exemplaires;
   float *prix;
} t_livre;
```

2 Affichage & saisie

Exercice 2

```
void affiche_livre(t_livre livre)
{  int i;
   printf("%s - ",livre.code);
   affiche_date(livre.publication);
   printf(" - %d exemplaires :",livre.exemplaires);
   for(i=0;i<livre.exemplaires;i++)
      printf(" %.2f",livre.prix[i]);
}
```

Exercice 3

```
void saisis_livre(t_livre *livre)
{  int i;
   printf("entrez le code : ");
   gets(livre->code);
   printf("entrez la date de publication : \n");
   saisis_date(&(livre->publication));
   printf("entrez le nombre d'exemplaires : ");
   scanf("%d",&(livre->exemplaires));
   if((livre->prix = (float*) malloc(sizeof(float)*livre->exemplaires))==NULL)
      printf("saisis_livre: erreur lors du malloc");
   else
   {  for(i=0;i<livre->exemplaires;i++)
      {  printf("entrez le prix de livre'exemplaire %d :",i+1);
         scanf("%f",&(livre->prix[i]));
      }
   }
}
```

3 Enregistrement & chargement

Exercice 4

```
void sauve_livre(t_livre livre, FILE *f)
{  int i;
   // code
   if(fprintf(f,"%s\t",livre.code) == EOF)
```

```
      {  printf("sauve_livre: erreur dans le fprintf du code\n");
         return;
      }
      // date
      sauve_date(livre.publication,f);
      // exemplaires
      if(fprintf(f,"\t%d\t",livre.exemplaires) == EOF)
      {  printf("sauve_livre: erreur dans le fprintf des exemplaires\n");
         return;
      }
      // prix
      for(i=0;i<livre.exemplaires;i++)
      {  if(fprintf(f,"%f\t",livre.prix[i]) == EOF)
         {  printf("sauve_livre: erreur dans le fprintf du prix n.%d\n",i);
            return;
         }
      }
   }
```

Exercice 5

```
   void charge_livre(t_livre *livre, FILE *fp)
   {  int i;
      // code
      if(fscanf(fp,"%s\t",livre->code)== EOF)
         printf("charge_livre: erreur dans le fscanf du code\n");
      // date
      else
      {  i = charge_date(&(livre->publication),fp);
         if(i!=-1)
         {  // exemplaires
            if(fscanf(fp,"\t%d\t",&(livre->exemplaires))== EOF)
               printf("charge_livre: erreur dans le fscanf des exemplaires\n");
            // prix
            else
            {  if((livre->prix = (float*) malloc(sizeof(float)*
                                                   livre->exemplaires))==NULL)
                  printf("charge_livre: erreur lors du malloc");
               for(i=0;i<livre->exemplaires;i++)
                  fscanf(fp,"%f\t",&(livre->prix[i]));
            }
         }
      }
   }
```

4 Stock de la librairie

Exercice 6

```
   void saisis_stock(t_livre stock[N], int* n)
   {  char reponse;
      *n = 0;
      do
      {  printf("Livre %d\n",(*n)+1);
         saisis_livre(&(stock[*n]));
         (*n) = (*n) + 1;
         if(*n<N)
         {  printf("Voulez-vous saisir un autre livre (O/N) ? ");
            scanf("%c",&reponse);
         }
         else
         {  printf("Impossible de saisir un autre livre : le stock est plein\n");
            reponse = 'N';
         }
      }
      while(reponse!='n' && reponse!='N');
      printf("Termine : %d livres ont été saisis\n",*n);
   }
```

Exercice 7

```
void affiche_stock(t_livre stock[N], int n)
{  int i;
   for(i=0;i<n;i++)
   {  printf("%d.",i+1);
      affiche_livre(stock[i]);
      printf("\n");
   }
}
```

Exercice 8

```
void sauve_stock(t_livre stock[N], int n, char* fichier)
{  int i;
   FILE *fp;

   // ouverture du fichier
   if((fp = fopen(fichier, "w")) == NULL)
   {  printf("sauve_stock: erreur dans fopen\n");
      exit(EXIT_FAILURE);
   }

   // nombre de livres
   fprintf(fp,"%d\n",n);

   // livres
   for(i=0;i<n;i++)
   {  sauve_livre(stock[i],fp);
      fprintf(fp,"\n");
   }

   // fermeture du fichier
   if((fclose(fp)) == EOF)
   {  printf("sauve_stock: erreur dans fclose\n");
      exit(EXIT_FAILURE);
   }
}
```

Exercice 9

```
void charge_stock(t_livre stock[N], int* n, char* fichier)
{  int i;
   FILE *fp;

   // ouverture du fichier
   if((fp = fopen(fichier, "r")) == NULL)
   {  printf("charge_stock: erreur dans fopen\n");
      exit(EXIT_FAILURE);
   }

   // nombre de livres
   fscanf(fp,"%d\n",n);

   // livres
   for(i=0;i<*n;i++)
   {  charge_livre(&(stock[i]),fp);
      fscanf(fp,"\n");
   }

   // fermeture du fichier
   if((fclose(fp)) == EOF)
   {  printf("charge_stock: erreur dans fclose\n");
      exit(EXIT_FAILURE);
   }
}
```

5 Opérations sur le stock

Exercice 10

```
int compte_exemplaires(t_livre stock[N], int n)
{  int i,resultat=0;
   for(i=0;i<n;i++)
      resultat = resultat + stock[i].exemplaires;
   return resultat;
}
```

Exercice 11

```
float estime_stock(t_livre stock[N], int n)
{  int i,j;
   float resultat = 0;
   for(i=0;i<n;i++)
   {  for(j=0;j<stock[i].exemplaires;j++)
         resultat = resultat + stock[i].prix[j];
   }
   return resultat;
}
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 38 | automates cellulaires

2 Représentation

Exercice 2

```
void vide_automate(int automate[N][M])
{  int i,j;
   for(i=0;i<N;i++)
      for(j=0;j<M;j++)
         automate[i][j] = 0;
}
```

Exercice 3

```
void copie_automate(int automate1[N][M], int automate2[N][M])
{  int i,j;
   for(i=0;i<N;i++)
      for(j=0;j<M;j++)
         automate2[i][j] = automate1[i][j];
}
```

Exercice 4

```
void dessine_automate(int automate[N][M])
{  int i,j;
   int x_dim = FENETRE_LARGEUR / M;
   int y_dim = FENETRE_HAUTEUR / N;

   efface_fenetre();

   // dessine les cellules
   for(i=0;i<N;i++)
   {  for(j=0;j<M;j++)
      {  if(automate[i][j])
            remplis_rectangle(j*x_dim,i*y_dim,x_dim,y_dim,C_BLANC);
      }
   }

   // dessine le quadrillage
   for(i=0;i<N;i++)
      dessine_segment(0,i*y_dim,FENETRE_LARGEUR-1,i*y_dim,C_GRIS_CLAIR);
   for(j=0;j<M;j++)
      dessine_segment(j*x_dim,0,j*x_dim,FENETRE_HAUTEUR-1,C_GRIS_CLAIR);

   raffraichis_fenetre();
}
```

3 Simulation

Exercice 5

```
int compte_voisines_vivantes(int automate[N][M], int i, int j)
{  int resultat = 0;
   int k,l;
   for(k=-1;k<=1;k++)
   {  for(l=-1;l<=1;l++)
      {  if((k!=0 || l!=0)
```

```
                && i+k>=0 && i+k<N
                && j+l>=0 && j+l<M)
                //resultat = resultat + automate[(i+k+N)%N][(j+l+M)%M];
                resultat = resultat + automate[i+k][j+l];
         }
      }

      return resultat;
   }
```

Exercice 6

```
   int itere_automate(int automate[N][M])
   {  int resultat = 0;
      int i,j,vv;
      int automate0[N][M];

      copie_automate(automate,automate0);

      for(i=0;i<N;i++)
      {  for(j=0;j<M;j++)
         {  vv = compte_voisines_vivantes(automate0,i,j);
            if(automate0[i][j])
               if(vv==2 || vv==3)
                  automate[i][j] = 1;
               else
               {  automate[i][j] = 0;
                  resultat = 1;
               }
            else
            {  if(vv==3)
               {  automate[i][j] = 1;
                  resultat = 1;
               }
               else
                  automate[i][j] = 0;
            }
         }
      }

      return resultat;
   }
```

Exercice 7

```
   void simule_automate(int automate[N][M], int iterations)
   {  int it=0,res;
      while((iterations && it<iterations)
         || (!iterations && res))
      {  res = itere_automate(automate);
         dessine_automate(automate);
         attends_delai(DELAI);
      }
   }
```

Exercice 8

```
   int main(int argc, char** argv)
   {  initialise_fenetre("Automates cellulaires");

      // init automate vide
      int automate[N][M];
      vide_automate(automate);

      // bloc
      automate[5][5] = 1;
      automate[5][6] = 1;
      automate[6][5] = 1;
      automate[6][6] = 1;
```

```
    // vecteur
    automate[10][10] = 1;
    automate[10][11] = 1;
    automate[10][12] = 1;

    // dessine les structures
    dessine_automate(automate);
    attends_touche();

    // simule
    simule_automate(automate,10);
    attends_touche();

    return EXIT_SUCCESS;
}
```

4 Structures cellulaires

Exercice 9

```
int charge_structure(char* fichier, int** structure, int* lignes, int* colonnes)
{  int i,j;
   FILE *fp;

   // on ouvre le fichier
   if((fp = fopen(fichier, "r")) == NULL)
   {  printf("charge_structure: erreur dans fopen\n");
      return -1;
   }

   // on lit les dimensions
   if(fscanf(fp,"%d\t%d\n",lignes,colonnes)== EOF)
   {  printf("charge_structure: erreur dans fscanf\n");
      return -1;
   }

   // on alloue la matrice
   if((*structure = (int*)malloc((*lignes)*(*colonnes)*sizeof(int))) == NULL)
   {  printf("charge_structure: erreur dans malloc\n");
      return -1;
   }

   // on lit son contenu
   for(i=0;i<*lignes;i++)
   {  // d'abord chaque valeur de la ligne
      for(j=0;j<*colonnes;j++)
      {  if(fscanf(fp,"%d\t",(*structure)+i*(*colonnes)+j)== EOF)
         {  printf("charge_structure: erreur dans fscanf\n");
            return -1;
         }
      }
      // puis la fin de ligne
      if(fscanf(fp,"\n")== EOF)
      {  printf("charge_structure: erreur dans fscanf\n");
         return -1;
      }
   }

   // on ferme le fichier
   if((fclose(fp)) == EOF)
   {  // traitement de l'erreur
      printf("charge_structure: erreur dans fclose\n");
      return -1;
   }

   return 0;
```

```
    }
```

Exercice 10

```
   void  ajoute_structure(int  automate[N][M],  int*  structure,  int  lignes,  int
colonnes, int i, int j)
   {  int k,l;

      for(k=0;k<lignes;k++)
      {  for(l=0;l<colonnes;l++)
            automate[(i+k)%N][(j+l)%M] = *(structure+k*colonnes+l);
      }
   }
```

Exercice 11

Vitesses des vaisseaux fournis avec le sujet :

- Planeur : $1/4$
- LWSS : $1/2$
- HWSS : $1/2$

Quant à proposer un autre vaisseau plus rapide… cf. le Web !

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 39 | fonctions récursives

1 Chaînes de caractères

Exercice 1

```
int est_contenu(char *chaine, char c)
{  int resultat;
   if(chaine[0]=='\0')
      resultat = 0;
   else if(chaine[0]==c)
      resultat = 1;
   else
      resultat = est_contenu(chaine+1,c);
   return resultat;
}
```

Exercice 2

```
int compte_occurrences(char *chaine, char c)
{  int resultat;
   if(chaine[0]=='\0')
      resultat = 0;
   else if(chaine[0]==c)
      resultat = 1 + compte_occurrences(chaine+1,c);
   else
      resultat = compte_occurrences(chaine+1,c);
   return resultat;
}
```

Exercice 3

```
int calcule_longueur(char *chaine)
{  int resultat;
   if(chaine[0]=='\0')
      resultat = 0;
   else
      resultat = 1 + calcule_longueur(chaine+1);
   return resultat;
}
```

Exercice 4

```
int calcule_position(char *chaine, char c)
{  int resultat;
   if(chaine[0]=='\0')
      resultat = -1;
   else if(chaine[0]==c)
      resultat = 0;
   else
   {  resultat = calcule_position(chaine+1,c);
      if(resultat!=-1)
         resultat ++;
   }
   return resultat;
}
```

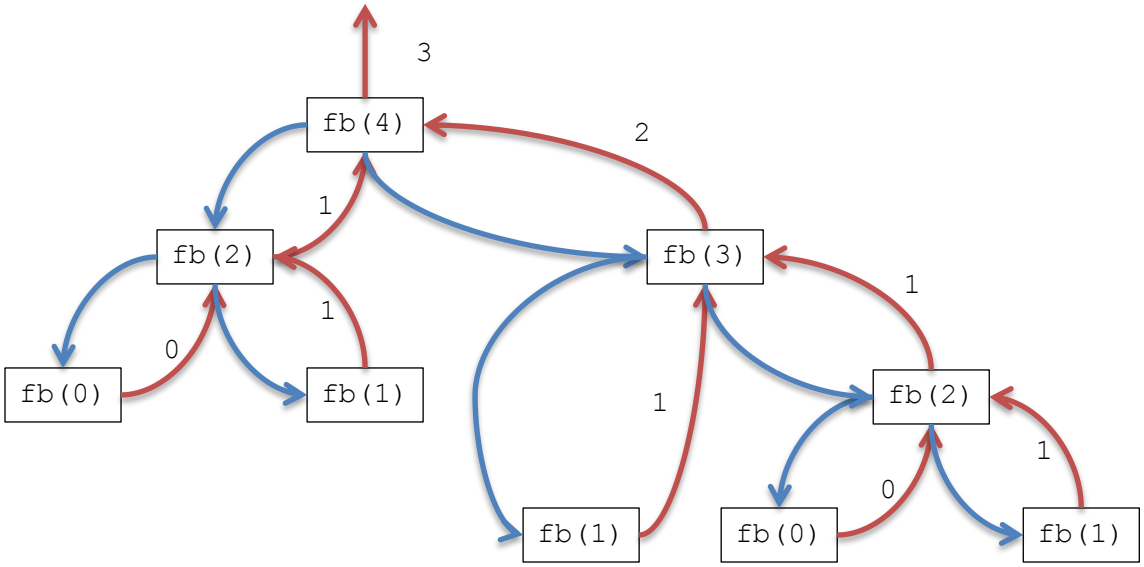
2 Entiers

Exercice 5

- Fonction `calcule_fibonacci` :

```
int calcule_fibonacci(int n)
{  int resultat;
   if(n==0)
      resultat = 0;
   else if(n==1)
      resultat = 1;
   else
      resultat = calcule_fibonacci(n-1) + calcule_fibonacci(n-2);
   return resultat;
}
```

- Arbre d'appels :



Il est évident que la fonction calcule plusieurs fois les mêmes valeurs : elle calcule deux fois u_2, par exemple. On pourrait accélérer le traitement en ne calculant qu'une seule fois chaque terme de la suite… ce type d'amélioration fera l'objet d'un TP à venir.

Exercice 6

```
int calcule_pgcd(int x, int y)
{  int resultat;
   if(y==0)
      resultat = x;
   else if(x==0)
      resultat = y;
   else if(x>=y)
      resultat = calcule_pgcd(x-y,y);
   else
      resultat = calcule_pgcd(x,y-x);
   return resultat;
}
```

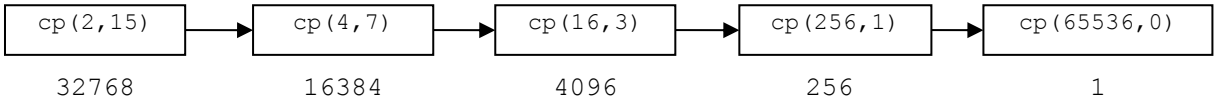
Exercice 7

- Fonction `calcule_puissance` :

```
long calcule_puissance(long x, long y)
{  long resultat;
   printf("appel : %ld^%ld\n",x,y);
   if (y==0)
      resultat = 1;
   else if ((y % 2) == 0)
      resultat = calcule_puissance(x*x,y/2);
```

```
    else
       resultat = x*calcule_puissance(x*x,y/2);
    return resultat;
}
```

- Arbre d'appels :



On peut observer que cette méthode est plus efficace que celle vue en cours, puisqu'elle réalise seulement 5 appels, alors que celle vue en cours en nécessiterait 15.

3 Chaînes de caractères (suite)

Exercice 8

- Calcul des chiffres romains :

```
int evalue_chiffre_romain(char chiffre)
{  int resultat=0;
   switch(chiffre)
   {  case 'M':
         resultat=1000;
         break;
      case 'D':
         resultat=500;
         break;
      case 'C':
         resultat=100;
         break;
      case 'L':
         resultat=50;
         break;
      case 'X':
         resultat=10;
         break;
      case 'V':
         resultat=5;
         break;
      case 'I':
         resultat=1;
   }
   return resultat;
}
```

- Calcul des nombres romains :

```
int evalue_nombre_romain(char *nombre)
{  int resultat;
   if(nombre[1]=='\0')
      resultat = evalue_chiffre_romain(nombre[0]);
   else
   {  int c0=evalue_chiffre_romain(nombre[0]);
      int c1=evalue_chiffre_romain(nombre[1]);
      if(c0>=c1)
         resultat = c0+evalue_nombre_romain(nombre+1);
      else
         resultat = evalue_nombre_romain(nombre+1)-c0;
   }
   return resultat;
}
```

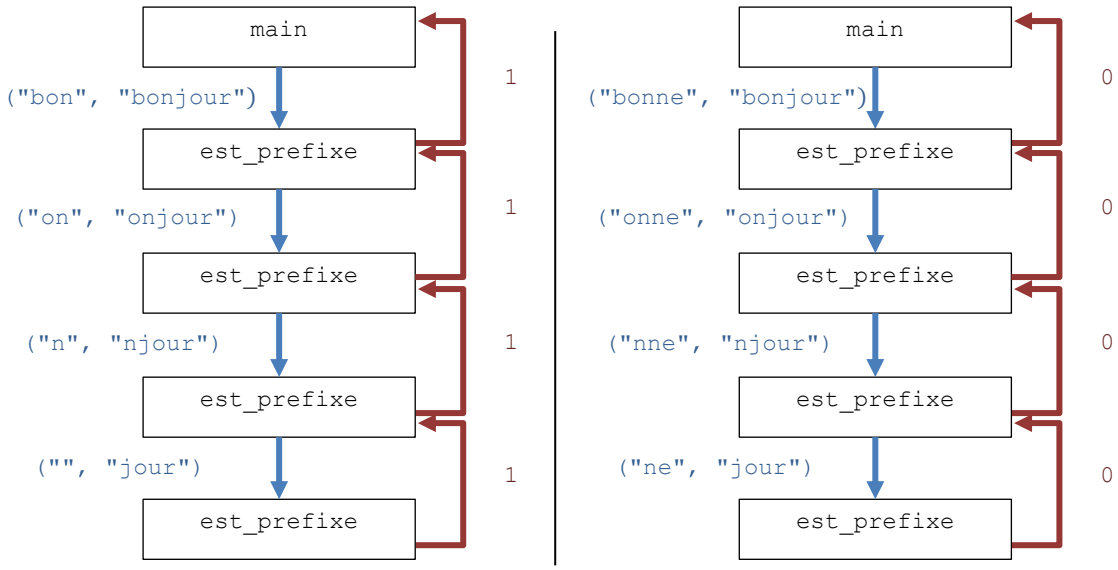
Exercice 9

- Fonction `est_prefixe` :

```
int est_prefixe(char *chaine1, char *chaine2)
{  int resultat;
```

```
    if(*chaine1=='\0')
       resultat = 1;
    else
    {  if(*chaine1 != *chaine2)
          resultat = 0;
       else
          resultat = est_prefixe(chaine1+1, chaine2+1);
    }

    return resultat;
}
```

- Arbres des appels :

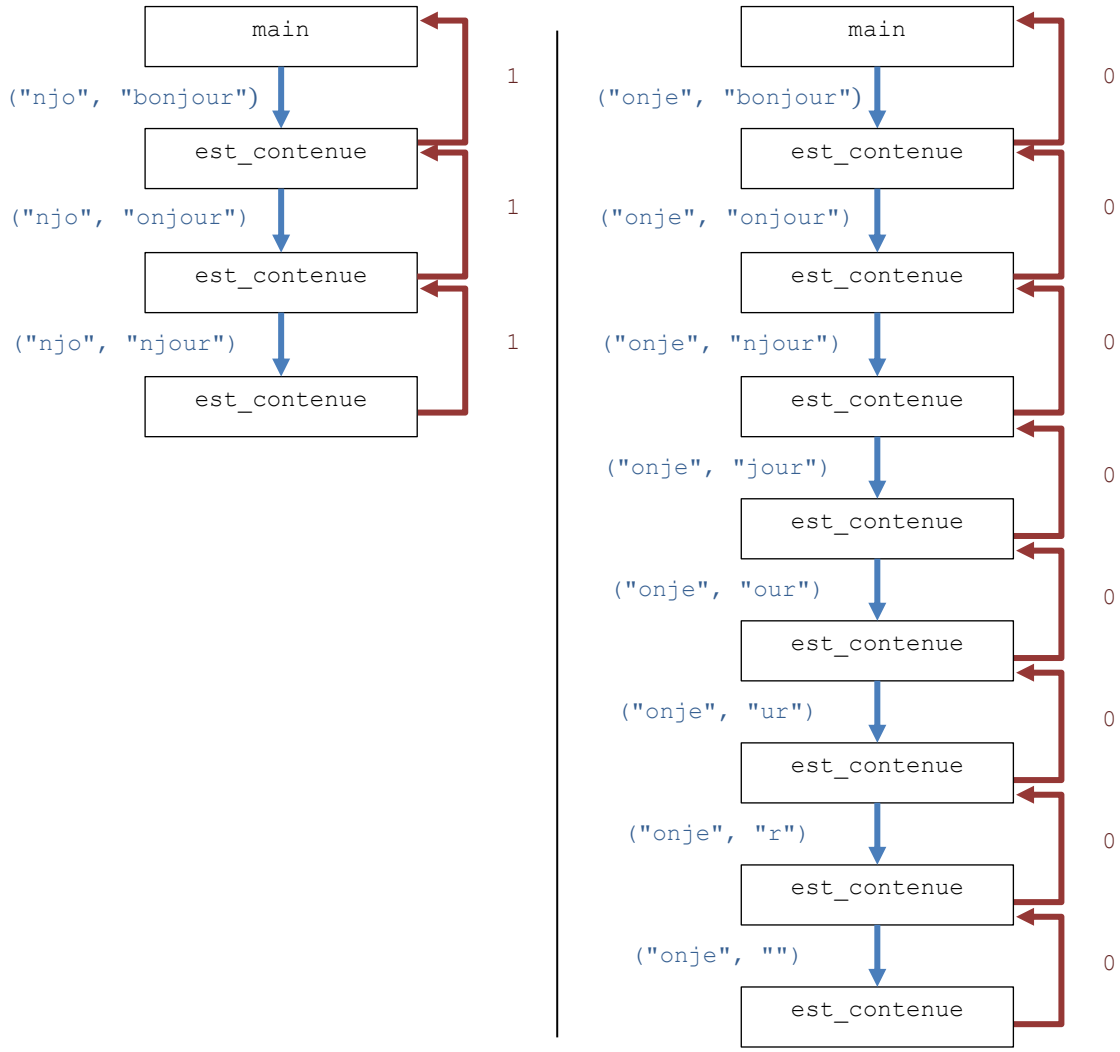


Exercice 10

- Fonction `est_contenue` :

```
int est_contenue(char* chaine1, char *chaine2)
{  int resultat;

   if(*chaine1!='\0' && *chaine2=='\0')
      resultat = 0;
   else
   {  if(est_prefixe(chaine1,chaine2))
         resultat = 1;
      else
         resultat = est_contenue(chaine1, chaine2+1);
   }

   return resultat;
}
```

- Arbres des appels :



Exercice 11

```
void ftnirp(char *chaine)
{  if(*chaine!='\0')
   {  ftnirp(chaine+1);
      putchar(*chaine);
   }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 40 | approximations numériques

1 Développement en série entière

Exercice 1

```
int calcule_factorielle(int n)
{  int resultat = 1;
   if(n>0)
      resultat = n * calcule_factorielle(n-1);
   return resultat;
}
```

Exercice 2

```
double calcule_puissance(double x, int n)
{  double resultat;
   if(n==0)
      resultat = 1;
   else if(n%2)
      resultat = x*calcule_puissance(x*x,n/2);
   else
      resultat = calcule_puissance(x*x,n/2);
   return resultat;
}
```

Exercice 3

```
double calcule_cosinus(double theta, double delta)
{  int i,signe;
   double resultat,terme;

   // init
   i = 0;
   resultat = calcule_puissance(theta,2*i)/calcule_factorielle(2*i);
   printf("i=%d\t%.10f\n",i,resultat);
   i++;
   signe = -1;

   do
   {  printf("i=%d\t%.10f ",i,resultat);
      terme = calcule_puissance(theta,2*i)/calcule_factorielle(2*i);
      if(signe==-1)
         printf("- %.10f = ",terme);
      else
         printf("+ %.10f = ",terme);
      resultat = resultat + signe*terme;
      printf("%.10f\n",resultat);
      signe = signe * -1;
      i++;
   }
   while(terme>=delta);

   return resultat;
}
```

2 Méthode de Monte-Carlo

Exercice 4

```
double tire_reel()
{  static int preums = 0;
   double resultat;

   // on initialise la graine si necessaire
   if(!preums)
   {  preums = 1;
      srand(time(NULL));
   }

   // on tire le reel entre 0 et 1
   resultat = rand() / (double) RAND_MAX;
   return resultat;
}
```

Exercice 5

```
double calcule_pi(int nc)
{  int i;
   double x, y, resultat, compte=0;

   // on tire nc points
   for(i=0;i<nc;i++)
   {  x = tire_reel();
      y = tire_reel();
      if(x*x+y*y<1)
         compte++;
   }

   // on calcule pi
   resultat = 4 * compte/nc;
   return resultat;
}
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 41 | figures fractales

2 Flocon de von Koch

Exercice 1

Soient $\delta_x = x_E - x_A$ et $\delta_y = y_E - y_A$. Alors on a :

$$x_B = x_A + \delta_x/3 \text{ et } x_D = x_E - \delta_x/3$$
$$y_B = y_A + \delta_y/3 \text{ et } y_D = y_E - \delta_y/3$$

Pour les coordonnées de *C*, on effectue une translation pour se ramener d'un centre de rotation *B* à un centre de rotation $(0,0)$, puis on applique la formule de la rotation, et enfin on effectue la translation inverse :

$$x_c = (x_D - x_B)\cos(\pi/3) - (y_D - y_B)\sin(\pi/3) + x_B$$
$$y_c = (x_D - x_B)\sin(\pi/3) + (y_D - y_B)\cos(\pi/3) + y_B$$

Implémentation :

- Pour améliorer la lisibilité du programme, on introduit une fonction calculant l'arrondi d'un réel :

```
int arrondit(float x)
{  int resultat=(int)x;
   if(x-resultat>=0.5)
      resultat++;
   return resultat;
}
```

- Fonction de calcul :

```
void calcule_segment_vonkoch(int xa, int ya, int *xb, int *yb, int *xc, int *yc,
int *xd, int *yd, int xe, int ye)
{  int dx,dy;
   dx = xe-xa;
   dy = ye-ya;

   *xb = arrondit(xa + dx/3.0);
   *yb = arrondit(ya + dy/3.0);
   *xd = arrondit(xe - dx/3.0);
   *yd = arrondit(ye - dy/3.0);
   *xc = arrondit(((*xd-*xb)*cos(-PI/3))-((*yd-*yb)*sin(-PI/3))+*xb);
   *yc = arrondit(((*xd-*xb)*sin(-PI/3))+((*yd-*yb)*cos(-PI/3))+*yb);
}
```

Exercice 2

```
void teste_segment_vonkoch(int xa, int ya, int xe, int ye)
{  int xb,yb,xc,yc,xd,yd;
   calcule_segment_vonkoch(xa,ya,&xb,&yb,&xc,&yc,&xd,&yd,xe,ye);

   dessine_segment(xa,ya,xb,yb,C_JAUNE);
   dessine_segment(xb,yb,xc,yc,C_JAUNE);
   dessine_segment(xc,yc,xd,yd,C_JAUNE);
   dessine_segment(xd,yd,xe,ye,C_JAUNE);

   rafraichit();
}
```

Exercice 3

```
void trace_segment_vonkoch(int xa, int ya, int xe, int ye, int n)
{  int xb,yb,xc,yc,xd,yd;

   // cas d'arret
   if (n==1)
   {  // on trace simplement le segment
      dessine_segment(xa,ya,xe,ye,C_JAUNE);
      // on attend un peu avant de rafraichir, pour ralentir l'affichage
      attend_delai(DELAI);
      rafraichit();
   }
   // cas general
   else
   {  // on calcule les positions de B, C et D
      calcule_segment_vonkoch(xa,ya,&xb,&yb,&xc,&yc,&xd,&yd,xe,ye);
      // on traite [A,B]
      trace_segment_vonkoch(xa,ya,xb,yb,n-1);
      // on traite [B,C]
      trace_segment_vonkoch(xb,yb,xc,yc,n-1);
      // on traite [C,D]
      trace_segment_vonkoch(xc,yc,xd,yd,n-1);
      // on traite [D,E]
      trace_segment_vonkoch(xd,yd,xe,ye,n-1);
   }
}
```

Exercice 4

```
void trace_flocon_vonkoch(int n)
{    int x1=600,y1=400;
     int x2=200,y2=400;
     int x3=400,y3=arrondit(400-200*sqrt(3));

     trace_segment_vonkoch(x1, y1, x2, y2, n);
     trace_segment_vonkoch(x2, y2, x3, y3, n);
     trace_segment_vonkoch(x3, y3, x1, y1, n);
}
```

3 Fractale arborescente

Exercice 5

```
/** Angle de la premiere rotation */
#define ALPHA PI/5
/** Angle de la deuxieme rotation */
#define BETA -PI/3
/** Ratio de coupe */
#define RATIO 0.3

void calcule_segment_fractarbre(int xa, int ya, int* xb, int* yb, int* xc, int*
yc, int* xd, int* yd, int xe, int ye)
{  int dx = xe-xa;
   int dy = ye-ya;

   *xb = arrondit(xa + RATIO*dx);
   *yb = arrondit(ya + RATIO*dy);
   *xc = arrondit(((xe-*xb)*cos(ALPHA))-((ye-*yb)*sin(ALPHA))+*xb);
   *yc = arrondit(((xe-*xb)*sin(ALPHA))+((ye-*yb)*cos(ALPHA))+*yb);
   *xd = arrondit(((xe-*xb)*cos(BETA))-((ye-*yb)*sin(BETA))+*xb);
   *yd = arrondit(((xe-*xb)*sin(BETA))+((ye-*yb)*cos(BETA))+*yb);
}
```

Exercice 6

```
void teste_segment_fractarbre(int xa, int ya, int xe, int ye)
```

```
{  int xb,yb,xc,yc,xd,yd;
   calcule_segment_fractarbre(xa,ya,&xb,&yb,&xc,&yc,&xd,&yd,xe,ye);

   dessine_segment(xa,ya,xb,yb,C_OCRE);
   dessine_segment(xb,yb,xc,yc,C_VERT);
   dessine_segment(xb,yb,xd,yd,C_VERT);

   rafraichit();
}
```

Exercice 7

```
void trace_fractarbre(int xa, int ya, int xe, int ye, int n)
{  int xb,yb,xc,yc,xd,yd;

   // cas d'arret
   if (n==1)
   {  // on trace le segment en vert (pour faire les feuilles)
      dessine_segment(xa,ya,xe,ye,C_VERT);
      // on attend un peu avant de rafraichir, pour ralentir l'affichage
      attend_delai(DELAI);
      rafraichit();
   }
   // cas general
   else
   {  // on calcule les positions de B, C et D
      calcule_segment_fractarbre(xa,ya,&xb,&yb,&xc,&yc,&xd,&yd,xe,ye);
      // on trace [A,B] en marron (pour faire le bois)
      dessine_segment(xa,ya,xb,yb,C_OCRE);
      // on trace [B,C]
      trace_fractarbre(xb,yb,xc,yc,n-1);
      // on trace [B,D]
      trace_fractarbre(xb,yb,xd,yd,n-1);
   }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 42 | listes chaînées

1 Opérations sur une liste simple

Exercice 1

```
int remplis_liste(int tab[], int taille, liste_s *l)
{  int erreur = 0;
   element_s *e;
   int i = 0;

   l->debut = NULL;
   while(i<taille && !erreur)
   {  if((e = cree_element_s(tab[i])) == NULL)
         erreur = -1;
      else
         insere_element_s(l,e,i);
      i++;
   }
   return erreur;
}
```

Exercice 2

```
int calcule_longueur(liste_s l)
{  element_s *temp = l.debut;
   int resultat = 0;

   while(temp != NULL)
   {  resultat++;
      temp = temp->suivant;
   }

   return resultat;
}
```

Exercice 3

```
int recherche_valeur(liste_s l, int v)
{  int i = 0;
   int pos = -1;
   element_s *temp = l.debut;

   while(pos==-1 && temp!=NULL)
   {  if(temp->valeur == v)
         pos = i;
      else
      {  temp = temp->suivant;
         i++;
      }
   }
   return pos;
}
```

Exercice 4

```
void melange_liste(liste_s *l)
{  int moitie = calcule_longueur(*l)/2;
   element_s *temp, *debut;
```

```
    int i;

    // si la liste est de longueur 0 ou 1, il n'y a rien à faire
    if(moitie == 0)
       return ;

    // ..:: etape 1 ::..
    // on trouve le milieu de la liste
    temp = l->debut;
    for(i=1;i<moitie;i++)
       temp = temp->suivant;
    // on coupe la liste
    debut = temp->suivant;
    temp->suivant = NULL;

    // ..:: etape 2 ::..
    // on trouve la fin de la liste
    temp = debut;
    while(temp->suivant != NULL)
       temp = temp->suivant;
    // on relie l'ancienne fin à l'ancienne tête
    temp->suivant = l->debut;
    // on positionne la nouvelle tête
    l->debut = debut;
}
```

2 Opérations sur plusieurs listes simples

Exercice 5

```
void fusionne_listes(liste_s *l1, liste_s l2)
{  element_s *temp = l1->debut;

   // si la liste 1 est vide, la fusion est la liste 2
   if(temp == NULL)
      l1->debut = l2.debut;
   else
   {  // sinon on va à la fin de la liste 1
      while(temp->suivant != NULL)
         temp = temp->suivant;
      //et on rajoute la liste 2
      temp->suivant = l2.debut;
   }
}
```

Exercice 6

```
int est_prefixe(liste_s l1, liste_s l2)
{  element_s *temp1=l1.debut, *temp2=l2.debut;
   liste_s l1_bis, l2_bis;
   int resultat;

   if (temp2==NULL)
      resultat = 1;
   else
      if (temp1==NULL)
         resultat = 0;
      else
         if (temp1->valeur == temp2->valeur)
         {  l1_bis.debut = temp1->suivant;
            l2_bis.debut = temp2->suivant;
            resultat = est_prefixe(l1_bis,l2_bis);
         }
         else
            resultat = 0;
   return resultat;
}
```

3 Opérations sur une liste double

Exercice 7

```
int echange_elements(liste_d *l, int p, int q)
{  element_d *temp_av,*temp_ap,*e_p,*e_q;
   int resultat=0;

   e_p = accede_element_d(*l, p);
   e_q = accede_element_d(*l, q);
   if(e_p==NULL || e_q==NULL)
      resultat = -1;
   else
   {  // pointeurs sur e_p
      if(e_p->precedent == NULL)
         l->debut = e_q;
      else
         e_p->precedent->suivant = e_q;
      if(e_p->suivant == NULL)
         l->fin = e_q;
      else
         e_p->suivant->precedent = e_q;
      //pointeurs sur e_q
      if(e_q->precedent == NULL)
         l->debut = e_p;
      else
         e_q->precedent->suivant = e_p;
      if(e_q->suivant == NULL)
         l->fin = e_p;
      else
         e_q->suivant->precedent = e_p;
      // pointeurs de e_p
      temp_av = e_p->precedent;
      temp_ap = e_p->suivant;
      e_p->precedent = e_q->precedent;
      e_p->suivant = e_q->suivant;
      // pointeurs de e_q
      e_q->precedent = temp_av;
      e_q->suivant = temp_ap;
   }
   return resultat;
}
```

Exercice 8

```
int detache_element(liste_d *l, element_d *e)
{  int erreur = 0;

   // cas ou la liste/element est vide
   if(l->debut==NULL || e==NULL)
      erreur = -1;
   else
   {  // cas ou la liste a un seul element
      if(l->fin == l->debut)
      {  l->fin = NULL;
         l->debut=NULL;
      }
      // cas ou la liste a plusieurs  elements
      else
      {  // cas ou e est en debut de liste
         if(e == l->debut)
         {  l->debut = e->suivant;
            l->debut->precedent = NULL;
         }
         // cas ou e est en fin de liste
         else if(e == l->fin)
         {  l->fin = e->precedent;
            l->fin->suivant = NULL;
```

```
            }
            // cas ou e n'est ni en debut ni en fin de liste
            else
            {   e->precedent->suivant = e->suivant;
                e->suivant->precedent = e->precedent;
            }
         }

         // on reinitialise les pointeurs de e
         e->precedent =  NULL;
         e->suivant = NULL;
      }

      return erreur;
   }
```

Exercice 9

```
   void inverse_liste(liste_d *l)
   {  element_d* e;
      liste_d temp;

      temp.debut = NULL;
      temp.fin = NULL;

      while(l->debut!=NULL)
      {  e = l->debut;
         detache_element(l,e);
         insere_element_d(&temp,e,0);
      }
      l->debut = temp.debut;
      l->fin = temp.fin;
   }
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 43 | disques & guirlandes

1 Points

Exercice 1

```
typedef struct
{  int x;
   int y;
} t_point;
```

Exercice 2

```
void affiche_point(t_point p)
{  printf("(%d,%d)",p.x, p.y);
}
```

2 Disques

Exercice 3

```
typedef struct
{  t_point centre;
   int r;
} t_disque;
```

Exercice 4

```
void affiche_disque(t_disque d)
{  printf("(c : ");
   affiche_point(d.centre);
   printf(", r : %d)",d.rayon);
}
```

Exercice 5

- Fonction traitant le centre :

```
void modifie_centre(t_disque* d, int d_x, int d_y)
{  (d->centre).x = (d->centre).x + d_x;
   (d->centre).y = (d->centre).y + d_y;
}
```

- Fonction traitant le rayon :

```
void modifie_rayon(t_disque* d, int d_r)
{  d->rayon = d->rayon + d_r;
}
```

Exercice 6

```
t_disque genere_disque()
{  t_disque resultat;
   int max_r;

   static int premiere_fois = 1;
   if(premiere_fois)
   {  srand(time(NULL));
      premiere_fois = 0;
   }
```

```
    // centre
    resultat.centre.x = 1 + rand()%(FENETRE_LARGEUR-1);
    resultat.centre.y = 1 + rand()%(FENETRE_HAUTEUR-1);

    // rayon
    max_r = FENETRE_LARGEUR-resultat.centre.x;
    if(resultat.centre.x<max_r)
       max_r = resultat.centre.x;
    if(FENETRE_HAUTEUR-resultat.centre.y<max_r)
       max_r = FENETRE_HAUTEUR-resultat.centre.y;
    if(resultat.centre.y<max_r)
       max_r = resultat.centre.y;
    // pour eviter un rayon trop grand
    if(50<max_r)
       max_r = 50;

    resultat.rayon = 1 + rand()%(max_r-1);

    return resultat;
}
```

Exercice 7

```
void anime_disque()
{  t_disque d={{50,300},20};
   int d_c=10, d_r=2;
   int i;

   for(i=0;i<104;i++)
   {  modifie_centre(&d, d_c=5, 0);
      modifie_rayon(&d, d_r);
      dessine_disque(d.centre.x,d.centre.y,d.rayon,C_BLEU);
      rafraichis_fenetre();
      attends_delai(10);
      efface_fenetre();
   }
}
```

3 Guirlandes

Exercice 8

- Modification du type `element_s` :

```
typedef struct s_element_s
{  t_disque d;
   struct s_element_s *suivant;
} element_s;
```

- Modification de la fonction d'affichage :

```
void affiche_liste_s(liste_s l)
{  element_s *temp = l.debut;

   printf("{");
   while(temp != NULL)
   {  affiche_disque(temp->d);
      printf(" - ");
      temp = temp->suivant;
   }
   printf(" }\n");
}
```

- Modification de la fonction de création :

```
element_s* cree_element_s(t_disque d)
{  element_s *e;
   if((e = (element_s *) malloc(sizeof(element_s))) != NULL)
   {  e->d = d;
      e->suivant = NULL;
   }
```

```
    return e;
}
```

Exercice 9

```
int genere_guirlande(liste_s* l, int n)
{  int i=0,erreur;
   element_s *e,*fin=NULL;
   t_disque d;

   // on genere la liste
   while(erreur!=-1 && i<n)
   {  d = genere_disque();
      if((e=cree_element_s(d)) == NULL)
         erreur = -1;
      else
         erreur = insere_element_s(l,e,0);
      if(fin==NULL)
         fin = e;
      i++;
   }

   // on referme la liste
   if(erreur!=-1)
      fin->suivant = l->debut;

   return erreur;
}
```

Exercice 10

```
void dessine_guirlande(liste_s l, Uint32 coul)
{  element_s *e = l.debut;
   t_disque d;
   t_point c1,c2;

   do
   {  d = e->d;
      c1 = d.centre;
      dessine_disque(c1.x,c1.y,d.rayon,coul);
      c2 = e->suivant->d.centre;
      dessine_segment(c1.x,c1.y,c2.x,c2.y,coul);
      e = e->suivant;
   }
   while(e != l.debut);

   rafraichis_fenetre();
}
```

Exercice 11

```
void clignote_guirlande(liste_s l, Uint32 coul1, Uint32 coul2, int k)
{  element_s *e = l.debut;
   t_disque d;
   t_point c1,c2;
   int i,j;
   Uint32 couls[2]={coul1,coul2},temp;

   // on fait les k iterations
   for(j=0;j<k;j++)
   {  // on dessine toute la guirlande
      do
      {  // on dessine deux disques (un de chaque couleur)
         for(i=0;i<2;i++)
         {  d = e->d;
            c1 = d.centre;
            c2 = e->suivant->d.centre;
            dessine_segment(c1.x,c1.y,c2.x,c2.y,coul1);
            dessine_disque(c1.x,c1.y,d.rayon,couls[i]);
```

```
            e = e->suivant;
         }
      }
      while(e != l.debut);
      rafraichis_fenetre();

      // on echange les couleurs pour la prochaine iteration
      temp = couls[0];
      couls[0] = couls[1];
      couls[1] = temp;

      // on attend un peu entre deux iterations
      attends_delai(500);
   }
}
```

Exercice 12

```
void anime_guirlande(liste_s l, Uint32 coul, Uint32 couls[], int c, int k)
{  int i,j;
   element_s *p = l.debut,*e;
   t_disque d;
   t_point ctr;

   for(i=0;i<k;i++)
   {  // on dessine la guirlande normalement
      dessine_guirlande(l, coul);

      // on rajoute les disques de couleurs differentes
      e = p;
      for(j=0;j<c;j++)
      {  d = e->d;
         ctr = d.centre;
         dessine_disque(ctr.x,ctr.y,d.rayon,couls[j]);
         e = e->suivant;
      }

      // on met a jour l'element de depart
      p = p->suivant;

      // on attend un peu entre deux iterations
      rafraichis_fenetre();
      attends_touche();
      attends_delai(250);
   }
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 44 | listes de caractères

1 Préparation

Exercice 1

Travail déjà fait de nombreuses fois : il faut modifier le type `element_s` et les fonctions `affiche_liste_s`, `cree_element_s` et `initialise_liste_s`.

Exercice 2

La liste n'a pas besoin d'être passée par adresse, car ce n'est pas la structure liste elle-même que l'on va modifier, mais les valeurs de ses éléments. Or les éléments sont indépendants de cette structure et stockés dans le tas, ils ne sont donc pas locaux à cette fonction. Par conséquent, les modifications apportées sur les éléments dans cette fonction ne seront pas perdues lorsque celle-ci se terminera.

```
void convertis_liste(liste_s l)
{  element_s *e = l.debut;

   while(e != NULL)
   {  if(e->valeur>='a' && e->valeur<='z')
         e->valeur = e->valeur - 'a' + 'A';
      e = e->suivant;
   }
}
```

2 Lecture

Exercice 3

```
liste_s construis_liste(FILE *fp)
{  int c,longueur=0,erreur=0;
   element_s *e;
   liste_s resultat;
   resultat.debut = NULL;

   while((c=fgetc(fp))!=EOF && erreur!=-1)
   {  if((e = cree_element_s(c)) == NULL)
         erreur = -1;
      else
      {  erreur = insere_element_s(&resultat,e,longueur);
         longueur++;
      }
   }
   if(erreur==-1)
      printf("construis_liste: impossible de rajouter le caractere %c dans la
                                                                  liste.\n",c);
   else if(!feof(fp))
      printf("construis_liste: impossible de lire le fichier.\n");

   return resultat;
}
```

Cette approche n'est pas efficace, car pour insérer chaque nouvel élément à la fin de la liste, il va falloir la parcourir intégralement à chaque fois. Il serait préférable d'avoir un pointeur courant sur le dernier élément.

Exercice 4

```
liste_s construis_liste2(FILE *fp)
{  int c,erreur=0;
   element_s *e,*temp=NULL;
   liste_s resultat;
   resultat.debut = NULL;

   while((c=fgetc(fp))!=EOF && erreur!=-1)
   {  if((e = cree_element_s(c)) == NULL)
         erreur = -1;
      else
      {  if(temp==NULL)
            resultat.debut = e;
         else
            temp->suivant = e;
         temp = e;
      }
   }
   if(erreur==-1)
      printf("construis_liste2: impossible de rajouter le caractere %c dans la
                                                                  liste.\n",c);
   else if(!feof(fp))
      printf("construis_liste2: impossible de lire le fichier.\n");

   return resultat;
}
```

Grâce à un pointeur courant sur le dernier élément de la liste, on peut directement insérer à la fin, sans avoir besoin de parcourir la liste intégralement à chaque insertion. Cette approche est donc plus efficace que la précédente.

3 Écriture

Exercice 5

```
void ecris_liste(liste_s l, FILE *fp)
{  element_s *e = l.debut;

   while(e!=NULL && fputc(e->valeur,fp)!=EOF)
      e = e->suivant;

   if(e!=NULL && !feof(fp))
      printf("ecris_liste: impossible d'ecrire le fichier.\n");
}
```

Exercice 6

```
void copie_fichier(char *source, char *cible)
{  FILE *fp;
   liste_s l;

   if((fp = fopen(source, "r")) == NULL)
   {  printf("copie_fichier: impossible d'ouvrir le fichier %s.\n",source);
   }
   else
   {  l = construis_liste2(fp);
      convertis_liste(l);
      if((fclose(fp)) == EOF)
      {  printf("copie_fichier: impossible de refermer le fichier
                                                          %s\n.",source);
      }
      else
      {  if((fp=fopen(cible,"w")) == NULL)
```

```
            {  printf("copie_fichier: impossible d'ouvrir le fichier %s.\n",cible);
            }
            else
            {  ecris_liste(l,fp);
               if((fclose(fp)) == EOF)
               {  printf("copie_fichier: impossible de refermer le fichier
                                                                    %s.\n",cible);
               }
            }
         }
      }
   }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 45 | tapis de Sierpiński

2 Tracé de carrés

Exercice 1

```
void trace_carre(point p, int cote)
{  int i,j;
   int x = p.x - cote/2;
   int y = p.y - cote/2;

   for(i=0;i<cote;i++)
   {  for(j=0;j<cote;j++)
      {  allume_pixel(x+i,y+j,p.coul);
      }
   }

   rafraichit();
   attend_delai(DELAI);
}
```

Exercice 2

```
void trace_carres(liste_s l, int cote)
{  element *e = l.debut;
   while(e!=NULL)
   {  trace_carre(e->p,cote);
      e = e->suivant;
   }
}
```

3 Version récursive

Exercice 3

```
int calcule_liste_centres(point p, int cote, liste_s *l)
{  int i=-cote,j,erreur=0;
   element *e;
   while(i<=cote && erreur!=-1)
   {  j = -cote;
      while(j<=cote && erreur!=-1)
      {  if(i!=0 || j!=0)
         {  point tmp;
            tmp.x = p.x + j;
            tmp.y = p.y + i;
            tmp.coul = p.coul;
            if((e=cree_element_s(tmp)) == NULL)
               erreur = -1;
            else
               erreur = insere_element_s(l,e,0);
         }
         j = j+cote;
      }
      i = i+cote;
   }
   return erreur;
}
```

Exercice 4

```
int trace_sierpinski_rec(point p, int cote)
{  int erreur = 0;
   element *e;
   liste_s l;
   l.debut = NULL;
   if(cote>0)
   {  // dessine le carre
      trace_carre(p,cote);
      // calcule les points suivants
      erreur = calcule_liste_centres(p,cote,&l);
      e = l.debut;
      while(e!=NULL && erreur!=-1)
      {  trace_sierpinski_rec(e->p,cote/3);
         e = e->suivant;
      }
   }
   return erreur;
}
```

4 Version itérative

Exercice 5

```
int calcule_liste_tous_centres(liste_s *l1, liste_s *l2, int cote)
{  int erreur = 0;
   element *e;
   point p;
   while(l1->debut!=NULL && erreur!=-1)
   {  e = l1->debut;
      p = e->p;
      l1->debut = e->suivant;
      free(e);
      erreur = calcule_liste_centres(p,cote,l2);
   }
   return erreur;
}
```

Exercice 6

```
int trace_sierpinski_it(point p, int cote)
{  int erreur=0;
   liste l1,l2;
   element *e;
   // initialisation
   l1.debut = NULL;
   l2.debut = NULL;
   if((e=cree_element_s(p)) == NULL)
      erreur = -1;
   else
      erreur = insere_element_s(&l1,e,0);
   // traitement
   while(cote>0 && erreur!=-1)
   {  // dessine les carres
      trace_carres(l1,cote);
      // prepare l'iteration suivante
      erreur = calcule_liste_tous_centres(&l1,&l2,cote);
      if(erreur != -1)
      {  l1.debut = l2.debut;
         l2.debut = NULL;
         cote = cote/3;
      }
   }
   return erreur;
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 46 | suite de Syracuse

1 Définition

Exercice 1

```
void insere_element_fin_d(liste_d *l, element_d *e)
{  element_d *avant = l->fin;

   // liste vide
   if(avant==NULL)
   {  l->debut = e;
      e->precedent = NULL;
   }
   // liste non-vide
   else
   {  avant->suivant = e;
      e->precedent = avant;
   }
   e->suivant = NULL;
   l->fin = e;
}
```

2 Implémentation itérative

Exercice 2

```
int calcule_syracuse_it1(int u0, liste_d *l)
{  int erreur=0, u=u0, fini=0;
   element_d *e;

   while(!fini && erreur!=-1)
   {  fini = u==1;
      if((e=cree_element_d(u))==NULL)
         erreur = -1;
      else
      {  insere_element_fin_d(l,e);
         if(u%2==0)
            u = u / 2;
         else
            u = 3*u + 1;
      }
   }

   return erreur;
}
```

Exercice 3

```
int calcule_syracuse_it2(int u0, liste_d *l, int *m, int *j)
{  int erreur=0, u=u0, fini=0;
   *m = u0;
   *j = 0;
   element_d *e;

   while(!fini && erreur!=-1)
   {  if(u==1)
         fini = 1;
```

```
        else
            (*j)++;
        if(u>*m)
            *m = u;
        if((e=cree_element_d(u))==NULL)
            erreur = -1;
        else
        {   insere_element_fin_d(l,e);
            if(u%2==0)
                u = u / 2;
            else
                u = 3*u + 1;
        }
    }

    return erreur;
}
```

3 Implémentation récursive

Exercice 4

- Fonction récursive :

```
int calcule_syracuse_rec1(liste_d *l)
{   int erreur=0, u;
    element_d *e=l->fin;
    if(e==NULL)
        erreur = -1;
    else
    {   u = e->valeur;
        if(u!=1)
        {   if(u%2==0)
                u = u / 2;
            else
                u = 3*u + 1;
            if((e=cree_element_d(u))==NULL)
                erreur = -1;
            else
            {   insere_element_fin_d(l,e);
                erreur = calcule_syracuse_rec1(l);
            }
        }
    }

    return erreur;
}
```

- Fonction d'initialisation :

```
int calcule_syracuse_rec1_init(int u0, liste_d *l)
{   int erreur=0;
    element_d *e;
    if((e=cree_element_d(u0))==NULL)
        erreur = -1;
    else
    {   insere_element_fin_d(l,e);
        erreur = calcule_syracuse_rec1(l);
    }
    return erreur;
}
```

Exercice 5

- Fonction récursive :

```
int calcule_syracuse_rec2(liste_d *l, int *m, int *j)
{   int erreur=0, u;
    element_d *e=l->fin;
    if(e==NULL)
```

```
        erreur = -1;
    else
    {   u = e->valeur;
        if(u!=1)
        {   if(u%2==0)
                u = u / 2;
            else
                u = 3*u + 1;
            if((e=cree_element_d(u))==NULL)
                erreur = -1;
            else
            {   (*j)++;
                if(u>*m)
                    *m = u;
                insere_element_fin_d(l,e);
                erreur = calcule_syracuse_rec2(l,m,j);
            }
        }
    }

    return erreur;
}
```

- Fonction d'initialisation :

```
int calcule_syracuse_rec2_init(int u0, liste_d *l, int *m, int *j)
{   int erreur=0;
    element_d *e;
    if((e=cree_element_d(u0))==NULL)
        erreur = -1;
    else
    {   *m = u0;
        *j = 0;
        insere_element_fin_d(l,e);
        erreur = calcule_syracuse_rec2(l,m,j);
    }
    return erreur;
}
```

4 Représentation graphique

Exercice 6

```
int vide_liste_d(liste_d *l)
{   int erreur=0;
    while(erreur!=-1 && l->debut!=NULL)
        erreur = supprime_element_d(l,0);
    return erreur;
}
```

Exercice 7

```
int calcule_temps(liste_d *l, int k)
{   int u0=2,erreur=0,j,m;
    element_d *e;
    liste_d t;

    t.debut = NULL;
    t.fin = NULL;

    while(erreur!=-1 && u0<=k)
    {   erreur = calcule_syracuse_it2(u0, &t, &m, &j);
        if(erreur!=-1)
        {   erreur = vide_liste_d(&t);
            if(erreur!=-1)
            {   e = cree_element_d(j);
                if(e==NULL)
                    erreur = -1;
                else
```

```
                {  insere_element_fin_d(l,e);
                   u0++;
                }
             }
          }
       }

       return erreur;
    }
```

Exercice 8

```
    void dessine_temps(liste_d l)
    {  int x_max = 1 + calcule_taille(l);
       element_d *e = trouve_max(l);
       int y_max = e->valeur;
       int x_marge = (FENETRE_LARGEUR-x_max)/2;
       int y_marge = (FENETRE_HAUTEUR-y_max)/2;
       int x=x_marge+2,y;
       int trait=2;

       // on dessine les points
       e = l.debut;
       while(e!=NULL)
       {  y = y_marge+y_max-e->valeur;
          dessine_point(x,y,C_ROUGE);
          e = e->suivant;
          x++;
       }

       // on dessine le repere
       dessine_segment(x_marge,y_marge,x_marge,y_marge+y_max,C_BLANC);
       dessine_segment(x_marge,FENETRE_HAUTEUR-y_marge,
                                  x_marge+x_max,FENETRE_HAUTEUR-y_marge,C_BLANC);
       // axe des abscisses
       for(x=0;x<x_max;x=x+10)
          dessine_segment(x_marge+x,FENETRE_HAUTEUR-y_marge,
                                  x_marge+x,FENETRE_HAUTEUR-y_marge+trait,C_BLANC);
       // axe des ordonnées
       for(y=0;y<y_max;y=y+10)
          dessine_segment(x_marge-trait,FENETRE_HAUTEUR-y_marge-y,x_marge,
                                              FENETRE_HAUTEUR-y_marge-y,C_BLANC);

       rafraichis_fenetre();
    }
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 47 | enveloppe d'un nuage de points

4 Implémentation

Exercice 1

- Structure de données `element_d` :

```
typedef struct s_element_d
{  int x;
   int y;
   float angle;
   struct s_element_d *precedent;
   struct s_element_d *suivant;
} element_d;
```

- Fonction `cree_element_d` :

```
element_d* cree_element_d(int x, int y, float angle)
{  element_d *e;
   if((e = (element_d *) malloc(sizeof(element_d))) != NULL)
   {  e->x = x;
      e->y = y;
      e->angle = angle;
      e->precedent = NULL;
      e->suivant = NULL;
   }
   return e;
}
```

Exercice 2

```
void genere_nuage(int n, liste_d *l)
{  element_d *e;
   int i,x,y;

   for(i=0;i<n;i++)
   {  genere_point(&x, &y);
      e = cree_element_d(x, y, 0);
      insere_element_d(l, e, 0);
   }
}
```

Exercice 3

```
void trace_nuage(liste_d l, Uint32 coul)
{  element_d *temp=l.debut;

   while(temp!=NULL)
   {  dessine_point(temp->x, temp->y, coul);
      temp = temp->suivant;
   }
}
```

Exercice 4

```
element_d* extrais_P0(liste_d *l)
{  element_d *temp=l->debut;
   element_d *p0=temp, *avant_p0, *apres_p0;

   if(temp!=NULL)
```

```
    {  while(temp->suivant!=NULL)
       {  temp = temp->suivant;
          if(temp->x<p0->x || (temp->x==p0->x && temp->y>p0->y))
             p0=temp;
       }
       avant_p0 = p0->precedent;
       apres_p0 = p0->suivant;
       if(avant_p0 == NULL)
          l->debut = apres_p0;
       else
          avant_p0->suivant = apres_p0;
       if(apres_p0 == NULL)
          l->fin = avant_p0;
       else
          apres_p0->precedent = avant_p0;
       p0->precedent = NULL;
       p0->suivant = NULL;
    }
    return p0;
}
```

Exercice 5

```
float calcule_angle(int v_x1, int v_y1, int v_x2, int v_y2)
{  float angle, produit_scalaire, cos_angle;
   float norme_v1, norme_v2, produit_normes;

   // cas particulier : les vecteurs sont les memes
   // (du au codage reel approximatif)
   if(v_x1==v_x2 && v_y1==v_y2)
      angle = 0;
   // cas general : les vecteurs sont differents
   else
   {  produit_scalaire = v_x1*v_x2 + v_y1*v_y2;
      norme_v1 = sqrt((v_x1*v_x1)+(v_y1*v_y1));
      norme_v2 = sqrt((v_x2*v_x2)+(v_y2*v_y2));
      produit_normes = norme_v1*norme_v2;
      // cas particulier pour eviter la division par zero
      if(produit_normes == 0)
         angle = 0;
      // cas general
      else
      {  cos_angle = produit_scalaire/produit_normes;
         angle = acos(cos_angle);
      }
   }
   return angle;
}
```

Exercice 6

```
void calcule_theta(liste_d *l, element_d p0)
{  element_d *temp=l->debut;
   int p0p_x, p0p_y, i_x=1, i_y=0;
   float theta;

   while(temp!=NULL)
   {  p0p_x = temp->x - p0.x;
      p0p_y = temp->y - p0.y;
           theta = calcule_angle(p0p_x, p0p_y, i_x, i_y);
      if(temp->y<p0.y)
         theta = -theta;
      temp->angle = theta;
        temp = temp->suivant;
   }
}
```

Exercice 7

```
int compare_elements_d(element_d e1, element_d e2)
{  int resultat=0;
   if(e1.angle>e2.angle)
      resultat = 1;
   else if(e1.angle<e2.angle)
      resultat = -1;
   else // si les thetas sont égaux
   {  if(e1.y<e2.y)
         resultat = -1;
      else
         resultat = 1;
   }
   return resultat;
}
```

Exercice 8

```
void calcule_enveloppe(liste_d *l)
{  element_d *p0;

   p0=extrais_P0(l);
   calcule_theta(l,*p0);
   trie_liste_decrois_d(l);
   insere_element_d(l,p0,0);
}
```

Exercice 9

```
void trace_polygone(liste l, Uint32 coul)
{  element_d *temp=l.debut, *temp2, *p0=temp;

   while(temp!=NULL)
   {  temp2 = temp->suivant;
      if(temp2 != NULL)
         dessine_segment(temp->x, temp->y, temp2->x, temp2->y, coul);
      else
         dessine_segment(temp->x, temp->y, p0->x, p0->y, coul);
      rafraichis_ecran();
      attends_delai(DELAI);
      temp = temp2;
   }
}
```

Exercice 10

```
   liste_d l;
   //element_d *p0;
   l.debut=NULL;
   l.fin=NULL;

    initialise_fenetre("Enveloppe d'un nuage de points");
   // initialiser le nuage de points
   genere_nuage(20, &l);
   // afficher le nuage de points en jaune
   trace_nuage(l, C_JAUNE);
   // calculer l'enveloppe quelconque
    calcule_enveloppe(&l);
   // afficher l'enveloppe quelconque en rouge
   trace_polygone(l, C_ROUGE);

   attends_touche();
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 48 | marches de Graham & Jarvis

3 Marche de Graham

Exercice 2

```
float calcule_determinant(int v_x1, int v_y1, int v_x2, int v_y2)
{  float resultat = v_x1*v_y2 - v_y1*v_x2;
   return resultat;
}
```

Exercice 3

```
void marche_graham(liste_d_d *l)
{  element_d *p0=l->debut,*p1=p0->suivant,*p2=p1->suivant;
   float determinant;
   int v_x1,v_y1,v_x2,v_y2;

   while(p1!=l->debut)
   {  dessine_segment(p0->x, p0->y, p1->x, p1->y, C_VERT);
      dessine_segment(p1->x, p1->y, p2->x, p2->y, C_VERT);
      rafraichis_ecran();
      attends_delai(DELAI);
      v_x1 = p0->x - p1->x;
      v_y1 = p0->y - p1->y;
      v_x2 = p2->x - p1->x;
      v_y2 = p2->y - p1->y;
      determinant = calcule_determinant(v_x1,v_y1,v_x2,v_y2);
      if(determinant<0)
      {  dessine_segment(p0->x, p0->y, p1->x, p1->y, C_NOIR);
         dessine_segment(p1->x, p1->y, p2->x, p2->y, C_NOIR);
         rafraichis_ecran();
         attends_delai(DELAI);
         p0->suivant = p2;
         p2->precedent = p0;
         free(p1);
         if(p0!=l->debut)
         {  p1 = p0;
            p0 = p0->precedent;
         }
      }
      else
      {  dessine_segment(p0->x, p0->y, p1->x, p1->y, C_BLEU);
         dessine_segment(p1->x, p1->y, p2->x, p2->y, C_NOIR);
         rafraichis_ecran();
         attends_delai(DELAI);
         p0 = p1;
         p1 = p2;
         p2 = p2->suivant;
         if(p2 == NULL)
            p2 = l->debut;
      }
   }
   dessine_segment(p0->x, p0->y, p1->x, p1->y, C_BLEU);
   rafraichis_ecran();
}
```

Remarque : les lignes en gris sont optionnelles, elles permettent de visualiser le déroulement de l'algorithme.

4 Marche de Jarvis

Exercice 4

```
void calcule_gamma(liste_d *l, element_d p1, element_d p2)
{  element_d *temp=l->debut;
   int v_x1=p1.x-p2.x,v_y1=p1.y-p2.y;
   int v_x2,v_y2;
   float gamma;

   while(temp!=NULL)
   {  v_x2 = temp->x-p2.x;
      v_y2 = temp->y-p2.y;
      gamma = calcule_angle(v_x1, v_y1, v_x2, v_y2);
      temp->angle = gamma;
      temp = temp->suivant;
   }
}
```

Exercice 5

```
element_d* extrais_suivant(liste_d *l, element_d p2)
{  element_d *temp=l->debut;
   element_d *p3=temp, *avant_p3, *apres_p3;

   if(temp!=NULL)
   {  while(temp->suivant!=NULL)
      {  temp = temp->suivant;
         if(temp->angle>p3->angle  ||  (temp->angle==p3->angle  &&  abs(p2.x-temp-
>x)>abs(p2.x-p3->x)))
            p3=temp;
      }
      avant_p3 = p3->precedent;
      apres_p3 = p3->suivant;
      if(avant_p3 == NULL)
         l->debut = apres_p3;
      else
         avant_p3->suivant = apres_p3;
      if(apres_p3 == NULL)
         l->fin = avant_p3;
      else
         apres_p3->precedent = avant_p3;
      p3->precedent = NULL;
      p3->suivant = NULL;
   }
   return p3;
}
```

Exercice 6

```
element_d* clone_element(element_d e)
{  element_d *resultat;

   resultat = cree_element_d(e.x, e.y, e.angle);
   resultat->precedent = e.precedent;
   resultat->suivant = e.suivant;
   return resultat;
}
```

Exercice 7

```
liste_d marche_jarvis(liste_d *l)
{  int v_ix=0,v_iy=-1;
   element_d *p0,p_init,*clone_p0;
   element_d *fin,*temp=NULL,*avant_fin;
   liste_d l2;

   // calcul de P0 et de son clone
   p0=extrais_P0(l);
```

```
        clone_p0 = clone_element_d(*p0);
        insere_element_d(l,clone_p0,0);
        l2.debut = p0;
        // calcul de P1
        p_init.x = p0->x+v_ix;
        p_init.y= p0->y+v_iy;
        calcule_gamma(l,p_init,*p0);
        fin = extrais_suivant(l,*p0);
        p0->suivant = fin;
        fin->precedent = p0;
        avant_fin = p0;
        // calcul du reste de l'enveloppe
        while(temp != clone_p0)
        {  calcule_gamma(l,*avant_fin,*fin);
           temp = extrais_suivant(l,*fin);
           if(temp != clone_p0)
           {  fin->suivant = temp;
              temp->precedent = fin;
              avant_fin = fin;
              fin = temp;
           }
        }
        l2.fin = fin;
        return l2;
    }
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 49 | analyse d'expressions

3 Implémentation de la méthode

Exercice 1

```
int est_operateur(char c)
{  return (c=='+' || c=='-' || c=='/' || c=='*' || c=='%');
}
```

Exercice 2

```
int est_chiffre(char c)
{  return (c>='0' && c<='9');
}
```

Exercice 3

```
int calcule_entier(char *exp, int *pos)
{  int resultat = 0;

   while(est_chiffre(exp[*pos]))
   {  resultat = resultat*10+(exp[*pos]-'0');
      (*pos)++;
   }
   return resultat;
}
```

Exercice 4

```
int applique_operateur(char opt, int opd1, int opd2)
{  int result;

   switch(opt)
   {  case '+':
         result = opd1+opd2;
         break;
      case '-':
         result = opd1-opd2;
         break;
      case '/':
         result = opd1/opd2;
         break;
      case '*':
         result = opd1*opd2;
         break;
      case '%':
         result = opd1%opd2;
         break;
   }
   return result;
}
```

Exercice 5

```
int evalue_NPI(char *exp, int *resultat)
{  pile p = cree_pile();
   int calcul,operande1,operande2;
   int erreur=0,pos=0;
   char operateur;
```

```
   while(erreur!=-1 && exp[pos]!='\0')
   {  if(est_chiffre(exp[pos]))
      {  operande1 = calcule_entier(exp,&pos);
         erreur = empile(&p,operande1);
      }
      else if(est_operateur(exp[pos]))
      {  operateur = exp[pos];
         erreur = sommet(p,&operande2);
         if(erreur!=-1)
            erreur = depile(&p);
         if(erreur!=-1)
            erreur = sommet(p,&operande1);
         if(erreur!=-1)
            erreur = depile(&p);
         if(erreur!=-1)
         {  calcul = applique_operateur(operateur,operande1,operande2);
            erreur = empile(&p,calcul);
            pos++;
         }
      }
      else if(exp[pos]==' ')
      {  if(pos==0
            || !est_chiffre(exp[pos-1]) || !est_chiffre(exp[pos+1]))
            erreur=-1;
         else
            pos++;
      }
      else
         erreur = -1;
   }
   if(erreur!=-1)
      erreur = sommet(p,resultat);
   if(erreur!=-1)
      erreur = depile(&p);
   if(!est_pile_vide(p))
      erreur = -1;
   return erreur;
}
```

4 Gestion des erreurs

Exercice 6

```
void interface_NPI()
{  char expression[30];
   int resultat,erreur;

   printf("entrez l'expression NPI a evaluer : ");
   gets(expression);
   erreur = evalue_NPI(expression,&resultat);
   if(erreur==-1)
      printf("erreur lors de l'evaluation.");
   else
      printf("le resultat est : %d.\n",resultat);
}
```

Exercice 7

Les modifications sont indiquées en **gras**.

```
int evalue_NPI_2(char *exp, int *resultat)
{  pile p = cree_pile();
   int calcul,operande1,operande2;
   int erreur=0,pos=0;
   char operateur;

   while(erreur==CODE_OK && exp[pos] !='\0')
   {  if(est_chiffre(exp[pos]))
```

```
         {  operande1 = calcule_operande(exp,&pos);
            erreur = empile(&p,operande1);
         }
         else if(est_operateur(exp[pos]))
         {  operateur = exp[pos];
            {  erreur = sommet(p,&operande2);
               if(erreur!=CODE_OK)
                  erreur = CODE_MANQUE_OPERANDE;
               else
                  erreur = depile(&p);
            }
            if(erreur==CODE_OK)
            {  erreur = sommet(p,&operande1);
               if(erreur!=CODE_OK)
                  erreur = CODE_MANQUE_OPERANDE;
               else
                  erreur = depile(&p);
            }
            if(erreur==CODE_OK)
            {  calcul = applique_operateur(operateur,operande1,operande2);
               erreur = empile(&p,calcul);
               pos++;
            }
         }
         else if(exp[pos]==' ')
         {  if(pos==0 || !est_chiffre(exp[pos-1])||!est_chiffre(exp[pos+1]))
               erreur = CODE_ESPACE_MAL_PLACE;
            else
               pos++;
         }
         else
            erreur = CODE_CARACTERE_INTERDIT;
      }
      if(erreur==CODE_OK)
      {  erreur = sommet(p,resultat);
         if(erreur!=CODE_OK)
            erreur = CODE_EXPRESSION_VIDE;
      }
      if(erreur==CODE_OK)
         erreur = depile(&p);
      if(erreur==CODE_OK)
      {  if(!est_pile_vide(p))
            erreur = CODE_MANQUE_OPERATEUR;
      }
      return erreur;
   }
```

Exercice 8

```
   void interface_NPI_2()
   {  char expression[30];
      int resultat,erreur;

      printf("entrez l'expression NPI a evaluer : ");
      gets(expression);
      erreur = evalue_NPI_2(expression,&resultat);
      switch(erreur)
      {  case CODE_OK :
            printf("le resultat est : %d.\n",resultat);
            break;
         case CODE_PILE :
            printf("erreur : acces a la pile.\n");
            break;
         case CODE_MANQUE_OPERANDE :
            printf("erreur : un operande est manquant.\n");
            break;
         case CODE_MANQUE_OPERATEUR :
            printf("erreur : un operateur est manquant.\n");
```

```
            break;
         case CODE_ESPACE_MAL_PLACE :
            printf("erreur : un espace est mal place.\n");
            break;
         case CODE_CARACTERE_INTERDIT :
            printf("erreur : un caractere interdit est present.\n");
            break;
         case CODE_EXPRESSION_VIDE :
            printf("erreur : l'expression est vide.\n");
            break;
      }
   }
```

Exercice 9

- Seule la fin de la fonction `evalue_NPI_3` est modifiée :

```
int evalue_NPI_3(char *exp, int *resultat, int *position)
{  ...
   if(erreur==CODE_OK)
   {  if(!est_pile_vide(p))
         erreur = CODE_MANQUE_OPERATEUR;
   }
   if(erreur!=CODE_OK)
      *position = pos;
   return erreur;
}
```

- Dans `interface_NPI_3`, la même modification est apportée à tous les messages d'erreur (sauf le tout dernier) :

```
void interface_NPI_3()
{  char expression[30];
   int resultat,erreur,position;

   printf("entrez l'expression NPI a evaluer : ");
   gets(expression);
   erreur = evalue_NPI_3(expression,&resultat,&position);
   switch(erreur)
   {  case CODE_OK :
         printf("le resultat est : %d.\n",resultat);
         break;
      case CODE_PILE :
         printf("erreur : acces a la pile (pos.%d).\n",position);
         break;
      ...
      case CODE_EXPRESSION_VIDE :
         printf("erreur : l'expression est vide.\n");
         break;
    }
}
```

Exercice 10

```
void souligne_erreur(char *expression, int position)
{  int i;

   printf("\t%s\n\t",expression);
   for(i=0;i<=position;i++)
      printf("^");
   printf("\n");
}
```

Exercice 11

Il suffit d'utiliser la fonction `souligne_erreur` après les messages d'erreurs :

```
void interface_NPI_4()
{  ...
      case CODE_PILE :
         printf("erreur : acces a la pile (pos.%d).\n",position);
```

```
            souligne_erreur(expression, position);
            break;
      ...
         case CODE_EXPRESSION_VIDE :
            printf("erreur : l'expression est vide.\n");
            break;
       }
   }
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 50 | conversion d'expressions

2 Parenthésage

Exercice 1

```
void passe_entier(char *exp, int *pos)
{  do
      (*pos)++;
   while(est_chiffre(exp[*pos]));
}
```

Exercice 2

```
int est_correcte_rec(char *exp, int *pos)
{  int resultat=0;

   if(exp[*pos]=='(')
   {  resultat=1;
      // opérande 1
      {  (*pos)++;
         if(est_chiffre(exp[*pos]))
            passe_entier(exp, pos);
         else
            resultat = est_correcte_rec(exp,pos);
      }
      // opérateur
      if(resultat)
         resultat = est_operateur(exp[*pos]);
      // opérande 2
      if(resultat)
      {  (*pos)++;
         if(est_chiffre(exp[*pos]))
            passe_entier(exp, pos);
         else
            resultat = est_correcte_rec(exp,pos);
      }
      // parenthèse fermante
      if(resultat)
         resultat = exp[*pos]==')';
      if(resultat)
           (*pos)++;
   }
   return resultat;
}
```

Exercice 3

```
int est_correcte(char *exp)
{  int pos=0,resultat;

   resultat = est_correcte_rec(exp,&pos);
   if(resultat)
      resultat = exp[pos]=='\0';
   return resultat;
}
```

3 Conversion & évaluation

Exercice 4

```
int convertis_infixe_vers_npi(char *exp1, char *exp2)
{  int erreur=0,operateur;
   char *pt1=exp1,*pt2=exp2;
   pile p = cree_pile();

   while(*pt1!='\0' && erreur!=-1)
   {  if(est_chiffre(*pt1))
      {  *pt2 = *pt1;
         pt2++;
      }
      else if(est_operateur(*pt1))
      {  empile(&p,*pt1);
         if(est_chiffre(*(pt2-1)))
         {  *pt2 = ' ';
            pt2++;
         }
      }
      else if(*pt1==')')
      {  sommet(p,&operateur);
         depile(&p);
         *pt2 = operateur;
         pt2++;
      }
      else if(*pt1!='(')
         erreur = -1;
      pt1++;
   }
   *pt2 = '\0';
   if(!est_pile_vide(p))
      erreur = -1;
   return erreur;
}
```

Exercice 5

```
int evalue_infixe(char *expression, int *resultat, int *position)
{  char npi[30];
   int erreur;

   if(!est_correcte(expression))
      erreur = CODE_PAS_CPLTMT_PARENTHESEE;
   else
   {  if(convertis_infixe_vers_npi(expression,npi)!=0)
         erreur = CODE_PB_CONVERSION;
      else
         erreur = evalue_NPI(npi,resultat,position);
   }

   return erreur;
}
```

Exercice 6

```
void interface_infixe()
{  char expression[30];
   int resultat,erreur,position;

   printf("Entrez l'expression infixe a evaluer : ");
   gets(expression);
   erreur = evalue_infixe(expression,&resultat,&position);
   switch(erreur)
   {  case CODE_OK :
         printf("Le resultat est : %d.\n",resultat);
         break;
      case CODE_PILE :
```

```
            printf("Erreur : acces a la pile (pos.%d).\n",position);
            souligne_erreur(expression, position);
            break;
         case CODE_MANQUE_OPERANDE :
            printf("Erreur : un operande est manquant (pos.%d).\n",position);
            souligne_erreur(expression, position);
            break;
         case CODE_MANQUE_OPERATEUR :
            printf("Erreur : un operateur est manquant (pos.%d).\n",position);
            souligne_erreur(expression, position);
            break;
         case CODE_ESPACE_MAL_PLACE :
            printf("Erreur : un espace est mal place (pos.%d).\n",position);
            souligne_erreur(expression, position);
            break;
         case CODE_CARACTERE_INTERDIT :
            printf("Erreur : un caractere interdit est present
                                                    (pos.%d).\n",position);
            souligne_erreur(expression, position);
            break;
         case CODE_EXPRESSION_VIDE :
            printf("Erreur : l'expression est vide.\n");
            break;
         case CODE_PAS_CPLTMT_PARENTHESEE :
            printf("Erreur : l'expression n'est pas completement parenthesee.\n");
            break;
         case CODE_PB_CONVERSION :
            printf("Erreur : probleme lors de la conversion en expression NPI.\n");
            break;
      }
   }
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 51 | détection de palindromes

2 Par renversement

Exercice 1

```
char* inverse_chaine(char *chaine)
{  char *result;
   int longueur = 0,i;
   // on calcule d'abord la longueur de la chaine
   while(chaine[longueur]!='\0')
      longueur++;
   // on fait l'allocation
   if((result=(char*)malloc((longueur+1)*sizeof(short)))!=NULL)
   {  // si l'allocation a marché, on recopie en inversant
      for(i=0;i<longueur;i++)
         result[i] = chaine[longueur-i-1];
      result[longueur] = '\0';
   }
   return result;
}
```

Exercice 2

```
int compare_chaines(char *chaine1, char *chaine2)
{  int result = 0;
   if(chaine1[0]==chaine2[0])
   {  if(chaine1[0]=='\0')
         result = 1;
      else
         result = compare_chaines(chaine1+1,chaine2+1);
   }
   return result;
}
```

Exercice 3

```
int est_palindrome1(char *chaine)
{  int resultat = 0;
   char *chaine2 = inverse_chaine(chaine);
   if(chaine2!=NULL)
      resultat = compare_chaines(chaine,chaine2);
   return resultat;
}
```

3 Par double parcours

Exercice 4

```
int est_palindrome2_it(char *chaine)
{  int resultat=1, debut=0, fin=0;
   // on positionne fin sur le dernier caractère
   while(chaine[fin]!='\0')
      fin++;
   fin--;
   // traitement iteratif de la chaine
   while(debut<fin && resultat==1)
   {  resultat = chaine[debut] == chaine[fin];
```

```
        debut++;
        fin--;
    }
    return resultat;
}
```

Exercice 5

- Algorithme :
    - Cas d'erreur : aucun, ou à la rigueur on peut tester si les paramètres sont non-nuls.
    - Cas d'arrêt :
        - Quand le début et la fin se croisent (i.e. `debut>=fin`) : succès.
        - Quand les caractères de début et de fin sont différents : échec.
    - Cas général : si le caractère de début et celui de fin sont égaux, alors on continue sur le milieu de la chaine, i.e. le nouveau début est le caractère suivant `debut` et la nouvelle fin est le caractère précédent `fin`.
- Implémentation :

```
int est_palindrome2_rec_sec(char *debut, char *fin)
{  int resultat = 0;
   if(debut>=fin)
      resultat = 1;
   else if(*debut==*fin)
      resultat = est_palindrome2_rec_sec(debut+1,fin-1);
   return resultat;
}
```

Exercice 6

```
int est_palindrome2_rec(char *chaine)
{  int resultat;
   char *debut=chaine,*fin=chaine;
   // on positionne fin sur le dernier caractère
   while(*fin!='\0')
      fin++;
   fin--;
   // traitement récursif de la chaine
   resultat = est_palindrome2_rec_sec(debut,fin);
   return resultat;
}
```

4 Par empilement

Exercice 7

```
int initialise_pile(char *chaine, pile *p)
{  int resultat=0,i=0;
   while(chaine[i]!='\0' && resultat!=-1)
   {  if((resultat=empile(p,chaine[i]))!=-1)
         i++;
   }
   return resultat;
}
```

Exercice 8

```
int est_palindrome3(char *chaine)
{  int resultat=1,i=0;
   char s;
   pile p = cree_pile();
   initialise_pile(chaine,&p);
   while(sommet(p,&s)==0 && resultat)
   {  resultat = chaine[i]==s;
      i++;
      depile(&p);
```

```
    }
    return resultat;
}
```

5 Généralisation

Exercice 9

```
void nettoie_chaine(char *lecture, char *ecriture)
{  while(*lecture!='\0')
   {  if(*lecture>='a' && *lecture<='z')
      {  *ecriture = *lecture;
         ecriture++;
      }
      else if(*lecture>='A' && *lecture<='Z')
      {  *ecriture = *lecture - 'A' + 'a';
         ecriture++;
      }
      lecture++;
   }
   *ecriture = '\0';
}
```

Exercice 10

```
int est_palindrome_phrase(char *phrase)
{  int resultat;
   nettoie_chaine(phrase,phrase);
   resultat = est_palindrome1(phrase);
   return resultat;
}
```

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 52 | tours de Hanoï

3 Implémentation

Exercice 1

```
typedef enum
{  gauche=0,
   centre,
   droite
} position;
```

Exercice 2

```
typedef struct
{  position pos;
   pile disques;
} axe;
```

Exercice 3

```
int initialise_probleme(axe *a_g, axe *a_c, axe *a_d)
{  int i=N,resultat=0;

   a_g->pos = gauche;
   a_g->disques = cree_pile();
   while(i>0 && resultat==0)
   {  resultat = empile(&(a_g->disques),i);
      i--;
   }
   if(resultat==0)
   {  a_c->pos = centre;
      a_c->disques = cree_pile();
      a_d->pos = droite;
      a_d->disques = cree_pile();
   }
   return resultat;
}
```

Exercice 4

```
Uint32 quelle_couleur(int disque)
{  Uint32 resultat;
   switch(disque)
   {  case 1:
         resultat = C_BLEU_FONCE;
         break;
      case 2:
         resultat = C_VERT_FONCE;
         break;
      case 3:
         resultat = C_CYAN_FONCE;
         break;
      case 4:
         resultat = C_ROUGE_FONCE;
         break;
      case 5:
         resultat = C_MAGENTA_FONCE;
         break;
```

```
         case 6:
            resultat = C_OCRE;
            break;
         case 7:
            resultat = C_GRIS_CLAIR;
            break;
         case 8:
            resultat = C_BLEU;
            break;
         case 9:
            resultat = C_GRIS;
            break;
         case 10:
            resultat = C_VERT;
            break;
         case 11:
            resultat = C_ROUGE;
            break;
         case 12:
            resultat = C_CYAN;
            break;
         case 13:
            resultat = C_MAGENTA;
            break;
         case 14:
            resultat = C_JAUNE;
            break;
         default:
            resultat = C_BLANC;
       }
       return resultat;
   }
```

Exercice 5

```
   void trace_axe(axe a, int x, int y)
   {  Uint32 coul;
      pile p;
      int disque,i,largeur;

      dessine_rectangle(x-AXE_LARGEUR/2, y, AXE_LARGEUR, AXE_HAUTEUR, C_BLANC);
      p = cree_pile();
      while(!est_pile_vide(a.disques))
      {  sommet(a.disques,&disque);
         depile(&(a.disques));
         empile(&p,disque);
      }
      affiche_pile(p);
      i=1;
      while(!est_pile_vide(p))
      {  sommet(p,&disque);
         depile(&p);
         coul = quelle_couleur(disque);
         largeur = DISQUE_LARGEUR_MIN+disque*DISQUE_LARGEUR_COEF;
         dessine_rectangle(x-largeur/2,   y+AXE_HAUTEUR-i*DISQUE_HAUTEUR,   largeur,
DISQUE_HAUTEUR, coul);
         i++;
      }
   }
```

Exercice 6

```
   void trace_probleme(axe a, axe b, axe c)
   {  axe tab[3];
      int x[3];
      int y = (FENETRE_HAUTEUR-AXE_HAUTEUR)/2;
      int dx = FENETRE_LARGEUR/10;
      int i;
```

```
      x[gauche] = 2*dx;
      x[centre] = FENETRE_LARGEUR/2;
      x[droite] = FENETRE_LARGEUR - 2*dx;
      tab[a.pos]=a;
      tab[b.pos]=b;
      tab[c.pos]=c;
      for(i=gauche;i<=droite;i=i+1)
         trace_axe(tab[i],x[i],y);
   }
```

4 Résolution

Exercice 7

```
int deplace_disque(axe *a, axe *b)
{  int disque_a,disque_b,resultat=0;

   if((resultat = sommet(a->disques,&disque_a)) != -1)
   {  if(!est_pile_vide(b->disques))
      {  sommet(b->disques,&disque_b);
         if(disque_a>=disque_b)
            resultat = -1;
      }
      if(resultat!=-1)
      {  if((resultat = empile(&(b->disques),disque_a)) != -1)
            depile(&(a->disques));
      }
   }
   return resultat;
}
```

Exercice 8

```
void rafraichis_probleme(axe a, axe b, axe c)
{  efface_fenetre();
   trace_probleme(a, b, c);
   rafraichis_fenetre();
   attends_delai(DELAI);
}
```

Exercice 9

```
void resous_hanoi(int n, axe *a, axe *b, axe *c)
{  if(n==1)
   {  deplace_disque(a,b);
      rafraichis_probleme(*a,*b,*c);
   }
   else
   {  resous_hanoi(n-1,a,c,b);
      deplace_disque(a,b);
      rafraichis_probleme(*a,*b,*c);
      resous_hanoi(n-1,c,b,a);
   }
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 53 | remplissage de formes

2 Remplissage par frontière

Exercice 1

```
1  void remplissage_frontiere(int xg, int yg, Uint32 coul)
2  {   Uint32 c;
3
4      c=couleur_pixel(xg,yg);
5      if(c!=COULEUR_FRONTIERE && c!=coul)
6      {   allume_pixel(xg,yg,coul);
7          raffraichis_fenetre();
8          attends_delai(DELAI);
9          remplissage_frontiere(xg-1,yg,coul);
10         remplissage_frontiere(xg+1,yg,coul);
11         remplissage_frontiere(xg,yg-1,coul);
12         remplissage_frontiere(xg,yg+1,coul);
13     }
14 }
```

Exercice 2

Taille des données :

- Soit *n* le nombre de pixels à colorier.

Complexité temporelle :

- ligne `4` : opération élémentaire + `couleur_pixel` : $O(1)$.
- ligne `5` : `if`
  - condition : $O(1)$.
  - `allume_pixel`, `raffraichit` et `attend` : $O(1)$.
  - appels récursifs :
    - soit k_i $(0 < i \leq 4)$ le nombre de pixels coloriés par le $i^{ème}$ appel dans cette fonction.
    - on a donc : $n = 1 + k_1 + k_2 + k_3 + k_4$.
    - conséquent :
      - ligne `9` : $T(n-1)$.
      - ligne `10` : $T(n-1-k_1)$.
      - ligne `11` : $T(n-1-\sum_{i=1}^{2} k_i)$.
      - ligne `12` : $T(n-1-\sum_{i=1}^{3} k_i)$.
    - il y a *n* pixels à colorier, et la fonction colorie directement au maximum 1 pixel par appel. Donc il y aura au minimum *n* appels pour colorier la figure (au total).
    - ce sont les appels inutiles (i.e. les appels pour des cas où il n'est pas possible de colorier un pixel) qui vont faire augmenter la complexité temporelle.
    - par conséquent, le pire des cas est celui où un seul des quatre appels fait tout le travail de coloriage, car c'est dans ce cas-là qu'on aura un maximum d'appels inutiles.
    - supposons que le premier appel fasse tout le travail, on a alors :

        - $k_1 = n - 1$
        - $k_2 = k_3 = k_4 = 0$
    - d'où, dans le pire des cas :
        - ligne `9` : $T(n-1)$.
        - ligne `10` : $T(0)$.
        - ligne `10` : $T(0)$.
        - ligne `12` : $T(0)$.
- **total :**
    - cas d'arrêt $n = 0$: $T(0) = O(1)$.
    - cas général $n > 0$: $T(n) = O(1) + T(n-1)$.
- on résout la relation de récurrence (cf. formule du cours) : $T(n) = O(n)$.

Complexité spatiale :

- 4 variables simples : $O(1)$
- relation de récurrence :
  $\max\left(S(n-1), S(n-1-k_1), S(n-1-\sum_{i=1}^{2} k_i), S(n-1-\sum_{i=1}^{3} k_i)\right)$
- comme pour la complexité temporelle, le pire des cas est celui où l'arbre d'appels est linéaire.
- on a donc la même complexité : $S(n) = O(n)$.

3 Balayage horizontal

Exercice 3

Les modifications sont les suivantes :

- `liste_s` :

```
typedef struct s_element_s
{  int x;
   int y;
   struct s_element_s *suivant;
} element_s;

element_s* cree_element_s(int x, int y)
{  element_s *e;
   if((e = (element_s *) malloc(sizeof(element_s))) != NULL)
   {  e->x = x;
      e->y = y;
      e->suivant = NULL;
   }
   return e;
}

void affiche_liste_s(liste_s l)
{  element_s *temp = l.debut;

   printf("{");
   while(temp != NULL)
   {  printf(" (%d,%d)", temp->x, temp->y);
      temp = temp->suivant;
   }
   printf(" }\n");
}
```

- `pile_liste` :

```
int empile(pile *p, int x, int y)
{  int resultat;
   element_s *e;
   e = cree_element_s(x,y);
   if(e == NULL)
      resultat = -1;
```

```
    else
       resultat = insere_element_s(p, e, 0);
    return resultat;
}

int sommet(pile p,  int *x, int *y)
{  int resultat;
   element_s *e = accede_element_s(p, 0);
   if(e == NULL)
      resultat = -1;
   else
   {  resultat = 0;
      *x = e->x;
      *y = e->y;
   }
   return resultat;
}
```

Exercice 4

```
int debut_segment(int x, int y)
{  Uint32 c;
   int resultat = x;

   do
   {  resultat--;
      c = couleur_pixel(resultat,y);
    }
   while(c!=COULEUR_FRONTIERE);
   resultat++;
   return resultat;
}
```

Exercice 5

```
void colorie_segment(int xd, int yd, Uint32 coul)
{  Uint32 c;
   int x=xd;

   do
   {  allume_pixel(x,yd,coul);
      raffraichis_fenetre();
      attends_delai(DELAI);
      x++;
      c = couleur_pixel(x,yd);
    }
   while(c!=COULEUR_FRONTIERE);
}
```

Exercice 6

```
void empile_suivants(pile *p, int xd, int yd, int position, Uint32 coul)
{  Uint32 c;
   int x=xd,y=yd+position;
   int nouveau = 1;

   do
   {  c = couleur_pixel(x,y);
      if(c==COULEUR_FRONTIERE)
         nouveau = 1;
      else if(c!=coul && nouveau)
      {  empile(p,x,y);
         nouveau = 0;
      }
      x++;
      c = couleur_pixel(x,yd);
   }
   while(c!=COULEUR_FRONTIERE);
}
```

Exercice 7

```
1  void balayage_horizontal(int xg, int yg, Uint32 coul)
2  {   pile p;
3      int x=xg,y=yg,xd;
4
5      p = cree_pile();
6      empile(&p,x,y);
7      while(!est_pile_vide(p))
8      {   sommet(p,&x,&y);
9          depile(&p);
10         xd = debut_segment(x,y);
11         colorie_segment(xd,y,coul);
12         empile_suivants(&p, xd, y, -1, coul);
13         empile_suivants(&p, xd, y, +1, coul);
14     }
15 }
```

Exercice 8

Taille des données :

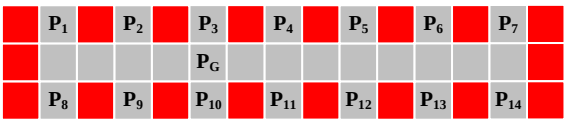
- soit n le nombre de pixels à colorier.
- on fait l'approximation suivante : $n = l \times h$ où l est la largeur de la figure et h sa hauteur.

Complexité temporelle :

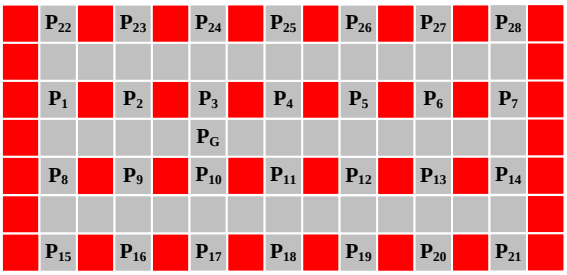
- lignes `3`, `5` et `6` : opération élémentaire + `cree_pile` + `empile` : $O(1)$.
- ligne `7` : `while`.
  - lignes `8` et `9` : `sommet` + `depile` : $O(1)$.
  - ligne `10` : affectation + `debut_segment` : $O(1) + O(l) = O(l)$.
    - la largeur maximale d'un segment est celle de la figure, c'est-à-dire l.
    - or, la fonction `debut_segment` est constituée d'une boucle `do/while` parcourant chaque pixel d'un segment, et effectuant une opération en temps constant, d'où sa complexité en $O(l)$.
  - lignes `11`, `12`, `13` : `colorie_segment` + 2 ×`empile_segment` : $O(l)$.
    - (pour les mêmes raisons que `debut_segment`).
  - nombre de répétions : h (une itération trace un segment horizontal, et il y a h segment horizontaux dans une figure de hauteur h).
  - total du `while` : $h \times O(l) = O(l \times h) = O(n)$.
- **total :** $T(n) = O(n)$

Complexité spatiale :

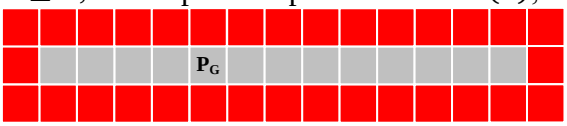
- 7 variables simples : $O(1)$.
- pile de données : trouver le pire des cas revient à répondre à la question : quel est le nombre d'éléments maximal dans la pile ?
  - soit un segment dont la largeur correspond à celle de la figure.
  - supposons qu'il y ait un maximum de segments au-dessus et au-dessous de ce segment.
  - chacun de ces segments fait un pixel de large, et est séparé du suivant par un pixel de frontière, donc il y a environ $l/2$ segments dessus et $l/2$ segments dessous.



- si la même structure se répète au-dessus et au-dessous sur toute la hauteur de la figure, on alors $(l/2) \times (h/2) = l \times h/4 = n/4$ segments dans la pile.



- **total :** $S(n) = O(n)$
- à noter que si $h \leq 3$, la complexité spatiale est en $O(1)$, car on n'utilise pas la pile.



- donc la complexité spatiale du balayage horizontal est inférieure à celle du remplissage par frontière, mais asymptotiquement on retrouve du $O(n)$ dans le pire des cas.
- on peut remarquer que ce pire des cas a peu de chances de se produire par rapport au pire des cas de l'algorithme de remplissage par frontière.
- **Interprétation :**
  - avec le remplissage par frontière, on empile des pixels, avec le balayage horizontal on empile des segments, d'où le gain de place.
  - *exemple :* pour colorier un polygone convexe, la pile contiendra au maximum 2 éléments : le segment situé au-dessus de la zone déjà coloriée, et le segment situé au-dessous.

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 54 | parcours d'un labyrinthe

2 Représentation

Exercice 1

```
typedef enum
{  mur,
   libre,
   courante,
   chemin,
   echec,
} valeur_case;
```

3 Résolution du problème

Exercice 2

```
int est_dans_labyrinthe(int x, int y)
{  return (x>=0 && x<DIM_LABY && y>=0 && y<DIM_LABY);
}
```

Exercice 3

```
int est_accessible(valeur_case laby[DIM_LABY][DIM_LABY], int x, int y)
{  return (laby[x][y]==libre && est_dans_labyrinthe(x,y));
}
```

Exercice 4

```
int est_impasse(valeur_case laby[DIM_LABY][DIM_LABY], int x, int y)
{  return (!est_accessible(laby,x,y-1)
      && !est_accessible(laby,x,y+1)
      && !est_accessible(laby,x-1,y)
      && !est_accessible(laby,x+1,y)
      && est_dans_labyrinthe(x,y));
}
```

Exercice 5

```
void recherche_largeur(valeur_case laby[DIM_LABY][DIM_LABY])
{  file f=cree_file();
   int xc=0,yc=0,fini=0;

   while((xc!=(DIM_LABY-1) || yc!=(DIM_LABY-1)) && !fini)
   {  if(est_impasse(laby,xc,yc))
         laby[xc][yc]=echec;
      else
         laby[xc][yc]=chemin;
      // case située au-dessus
      enfile(&f,xc,yc-1);
      // case située en-dessous
      enfile(&f,xc,yc+1);
      // case située à gauche
      enfile(&f,xc-1,yc);
      // case située à droite
      enfile(&f,xc+1,yc);
      // position courante
      do
```

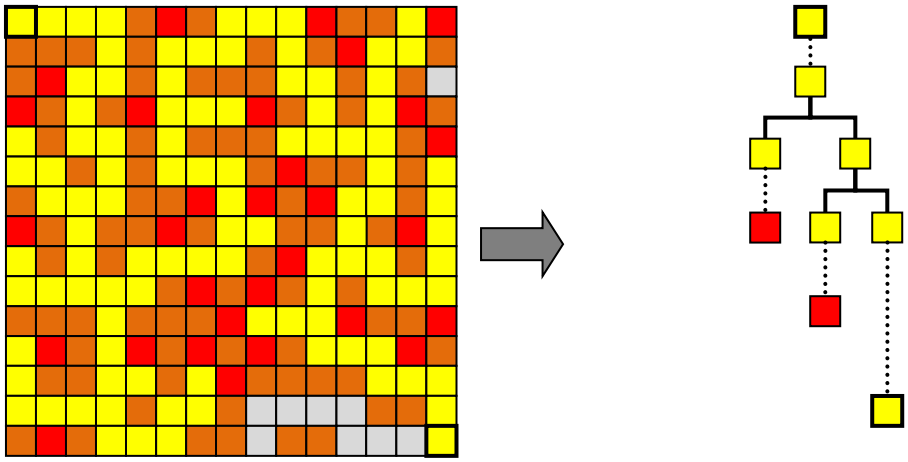
```
            {   tete(f,&xc,&yc);
                defile(&f);
            }
            while(!est_accessible(laby,xc,yc) && !est_file_vide(f));
            // affichage
            if(est_accessible(laby,xc,yc))
            {   laby[xc][yc]=courante;
                dessine_labyrinthe(laby);
                attends_delai(DELAI);
            }
            else
                fini = est_file_vide(f);
        }
    }
```

Exercice 6

Même fonction que pour la largeur, en utilisant une pile à la place d'une file.

Exercice 7

Les chemins développés par les algorithmes lors de la recherche de solution ont une structure d'arbre. La racine de l'arbre correspond à l'entrée du labyrinthe, les feuilles correspondent aux impasses (échecs) et à l'arrivée (succès).



Le parcours en profondeur développe une branche de l'arbre jusqu'à arriver à une feuille. Si cette feuille est la solution, l'algorithme est terminé. Si c'est un échec, l'algorithme revient au dernier embranchement et développe une branche sœur.

À la différence du parcours en profondeur, le parcours en largeur ne se concentre pas sur un seul chemin : il développe en parallèle tous les chemins possibles, jusqu'à ce qu'un des chemins arrive à la solution.

Comme l'illustrent les deux labyrinthes donnés en exemple, certains problèmes seront favorables à un parcours en profondeur, d'autres à un parcours en largeur.

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 55 | génération d'un labyrinthe

2 Méthode par fusion

Exercice 1

```
int initialise_grille(int pseudo[DIM_LABY][DIM_LABY])
{  int i,j;
   int n=0;

   for(i=0;i<DIM_LABY;i++)
      for(j=0;j<DIM_LABY;j++)
         if(i%2==0 && j%2==0)
         {  n++;
            pseudo[i][j] = n;
         }
         else
            pseudo[i][j] = 0;
   return n;
}
```

Exercice 2

```
int est_utile(int pseudo[DIM_LABY][DIM_LABY], int x, int y)
{  int v_x[4],v_y[4],val[4];
   int i,j,compte=0;

   // comparaison des zones
   v_x[0]=x;   v_y[0]=y-1; val[0]=pseudo[x][y-1];
   v_x[1]=x;   v_y[1]=y+1; val[1]=pseudo[x][y+1];
   v_x[2]=x-1; v_y[2]=y;   val[2]=pseudo[x-1][y];
   v_x[3]=x+1; v_y[3]=y;   val[3]=pseudo[x+1][y];

   // y a-t-il uniquement des murs dans les cases voisines ?
   i=0;
   do
   {  compte = !val[i];
      i++;
    }
   while(compte && i<4);

   // sinon : on compare les zones des cases voisines
   i=0;
   while(compte==0 && i<4)
   {  if(val[i] && est_dans_labyrinthe(v_x[i],v_y[i]))
      {  j=i+1;
         while(compte==0 && j<4)
         {  if(val[j]
               && est_dans_labyrinthe(v_x[j],v_y[j])
               && val[i]!=val[j])
               compte++;
            j++;
         }
      }
      i++;
   }
   return compte>0;
}
```

Exercice 3

```
void remplace_valeur(int pseudo[DIM_LABY][DIM_LABY], int ancienne, int nouvelle)
{  int i,j;

   for(i=0;i<DIM_LABY;i++)
      for(j=0;j<DIM_LABY;j++)
         if(pseudo[i][j] == ancienne)
            pseudo[i][j] = nouvelle;
}
```

Exercice 4

```
void supprime_mur(int pseudo[DIM_LABY][DIM_LABY], int x, int y, int *n)
{  int v_x[4],v_y[4],val[4];
   int i,min=*n+1;

   // recherche de la zone de plus petit numéro
   v_x[0]=x;  v_y[0]=y-1;val[0]=pseudo[x][y-1];
   v_x[1]=x;  v_y[1]=y+1;val[1]=pseudo[x][y+1];
   v_x[2]=x-1;v_y[2]=y;  val[2]=pseudo[x-1][y];
   v_x[3]=x+1;v_y[3]=y;  val[3]=pseudo[x+1][y];
   for(i=0;i<4;i++)
   {  if(est_dans_labyrinthe(v_x[i],v_y[i])
         && val[i] && val[i]<min)
         min = val[i];
   }

   // nouvelle zone ?
   if(min==*n+1)
   {  *n = *n+1;
      pseudo[x][y] = *n;
   }

   // fusion
   else
   {  for(i=0;i<4;i++)
      {  if(est_dans_labyrinthe(v_x[i],v_y[i]) && val[i])
            remplace_valeur(pseudo,val[i],min);
      }
      pseudo[x][y]=min;
   }
}
```

Exercice 5

```
int teste_zone_unique(int pseudo[DIM_LABY][DIM_LABY])
{  int i,j,resultat=1;

   i=0;
   while(i<DIM_LABY && resultat)
   {  j=0;
      while(j<DIM_LABY && resultat)
      {  resultat = (pseudo[i][j]==pseudo[0][0] || !pseudo[i][j]);
         j++;
      }
      i++;
    }
    return resultat;
}
```

Exercice 6

```
void genere_pseudo_labyrinthe(int pseudo[DIM_LABY][DIM_LABY])
{  int x,y,n;

   n = initialise_grille(pseudo);
    while(!teste_zone_unique(pseudo))
   {  do
         tire_mur(pseudo, &x, &y);
```

```
        while(!est_utile(pseudo,x,y));
        supprime_mur(pseudo,x,y,&n);
        dessine_pseudo_labyrinthe(pseudo);
        attends_delai(DELAI);
    }
    remplace_valeur(pseudo,pseudo[0][0],libre);
    dessine_pseudo_labyrinthe(pseudo);
}
```

Exercice 7

```
void convertis_labyrinthe(int pseudo[DIM_LABY][DIM_LABY],
                                        valeur_case laby[DIM_LABY][DIM_LABY])
{  int i,j;
   for(i=0;i<DIM_LABY;i++)
   {  for(j=0;j<DIM_LABY;j++)
      {  // case libre
         if(pseudo[i][j])
            laby[i][j] = libre;
         // case contenant un mur
         else
            laby[i][j] = mur;
      }
   }
}
```

Exercice 8

```
int main(int argc, char** argv)
{  int pseudo[DIM_LABY][DIM_LABY];
   valeur_case laby[DIM_LABY][DIM_LABY];

   initialise_fenetre("Generation de labyrinthes");
   genere_pseudo_labyrinthe(pseudo);
   convertis_labyrinthe(pseudo,laby);
   recherche_profondeur(laby);

   attends_touche();
   return 0;
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 56 | tri par dénombrement

1 Initialisation

Exercice 2

Première méthode :

Une méthode simple pour générer un entier compris entre *inf* et *sup* (bornes incluses) consiste d'abord à calculer l'écart existant ces deux bornes :

$$ecart = sup - inf + 1$$

La fonction `rand` renvoie un entier *x* tel que $x \in [0, RAND\_MAX]$. Il vient :

$$x \bmod ecart \in [0, ecart - 1]$$

Si on rajoute la borne inférieure, on obtient donc :

$$(inf + x \bmod ecart) \in [inf, sup]$$

Le problème de cette méthode est que si $RAND\_MAX$ n'est pas un multiple de *ecart*, on perd l'uniformité de la distribution (certaines valeurs apparaitront moins souvent que d'autres).

Deuxième méthode :

La fonction `rand` renvoie un entier *x* tel que $x \in [0, RAND\_MAX]$. Il vient :

$$\frac{x}{RAND\_MAX} \in [0,1]$$

On aura donc la valeur 1 si $x = RAND\_MAX$, et la valeur 0 pour toutes les autres valeurs de x, ce qui ne forme pas une distribution très uniforme. Il est plus intéressant d'utiliser la forme suivante :

$$\frac{x}{RAND\_MAX + 1} \in [0,1[$$

Il vient alors :

$$(sup - inf + 1)\frac{x}{RAND\_MAX + 1} \in [0, (sup - inf + 1)[$$

$$(sup - inf + 1)\frac{x}{RAND\_MAX + 1} + inf \in [inf, sup + 1[$$

Attention à l'implémentation : il est possible que la valeur $(sup - inf + 1) \times x$ provoque un dépassement de capacité suivant le type utilisé (`int`, `long`…). Pour éviter cela, on peut diviser par l'inverse de $(sup - inf + 1)$:

$$\frac{x}{\frac{RAND\_MAX + 1}{(sup - inf + 1)}} + inf \in [inf, sup + 1[$$

Cette formule est valide seulement si on n'utilise pas des divisions entières, sinon il est possible d'obtenir $\frac{x}{\frac{RAND\_MAX+1}{(sup-inf+1)}} = sup$, et donc une valeur finale égale à $sup + 1$. Pour remédier à cela, il faudra donc forcer l'utilisation des divisions réelles en effectuer une conversion explicite en réel lors de l'implémentation.

La fonction suivante est une implémentation de cette formule :

```
int genere_entier(int inf, int sup)
```

```
{  static int preums = 1;
   int resultat;

   if(preums)
   {  preums=0;
      srand(time(NULL));
   }
   resultat=rand()/(((float)RAND_MAX+1)/(sup-inf+1))+inf;
   return resultat;
}
```

Il faut utiliser une variable de classe de mémorisation `static` pour pouvoir initialiser le générateur pseudo-aléatoire lors du premier appel de la fonction.

Exercice 3

```
void affiche_tableau(int tab[], int taille)
{  int i;

   printf("{ ");
   for(i=0;i<taille;i++)
      printf("%d ",tab[i]);
   printf("}\n");
}
```

Exercice 4

```
void init_tableau(int tab[N], int n)
{  int i;

   for(i=0;i<N;i++)
      tab[i] = genere_entier(0,n);
}
```

Exercice 5

```
void init_tableau_unique(int tab[N], int n)
{  int i,j,present;

   for(i=0;i<N;i++)
   {  do
      {  tab[i] = genere_entier(0,n);
         present = 0;
         j = 0;
         while(j<i && !present)
         {  present = tab[i]==tab[j];
            j++;
         }
      }
      while(present);
   }
}
```

2 Algorithme de tri

Exercice 6

```
void calcule_denombrement(int tab1[N], int tab2[M])
{  int i;

   for(i=0;i<M;i++)
      tab2[i]=0;
   for(i=0;i<N;i++)
      tab2[tab1[i]]++;
}
```

Exercice 7

```
void calcule_distribution(int tab2[M], int tab3[M])
{  int i;
```

```
    tab3[0]=tab2[0];
    for(i=1;i<M;i++)
       tab3[i]=tab3[i-1]+tab2[i];
}
```

Exercice 8

```
void calcule_construction(int tab1[N], int tab3[M], int tab4[N])
{  int i,v;

   for(i=N-1;i>=0;i--)
   {  v = tab1[i];
      tab3[v]--;
      tab4[tab3[v]] = v;
   }
}
```

Exercice 9

```
void tri_denombrement(int tab1[N], int tab4[N])
{  int tab2[M],tab3[M];

   calcule_denombrement(tab1,tab2);
   calcule_distribution(tab2,tab3);
   calcule_construction(tab1,tab3,tab4);
}
```

Exercice 10

Complexité temporelle :

- `denombrement` : $O(N+M)$.
- `distribution` : $O(M)$.
- `construction` : $O(N)$.
- `tri_denombrement` : $O(N+M)$.

Propriétés :

- En place : **non** (on utilise plusieurs tableaux auxiliaire dont un a la même taille que la séquence initiale).
- Stabilité : **oui** (lors de la construction, le fait de partir de la fin du tableau permet de préserver les positions relatives des éléments de même clé).

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 57 | tri cocktail

1 Version itérative

Exercice 1

L'utilisation d'une fonction d'échange permet de simplifier la fonction de tri, car on doit considérer deux cas dans cette dernière : le sens de parcours du tableau (allez ou retour).

- la fonction `echange` :

```
   void echange(int* a, int* b)
1  {   int temp;
2      temp = *a;
3      *a=*b;
4      *b=temp;
   }
```

- la fonction de tri :

```
   void tri_cocktail_it(int tab[N])
0  {   int i=0,j,permut,deb,fin;

1      do
2      {  permut = 0;
3         deb = i/2;
4         fin = N-1 - deb;

          // allez
5         if(i%2==0)
6         {  for(j=deb;j<fin;j++)
7            {  if(tab[j]>tab[j+1])
8               {  echange(&tab[j],&tab[j+1]);
                   permut = 1;
                }
            }
          }

          // retour
10        else
11        {  for(j=fin;j>deb;j--)
12           {  if(tab[j]<tab[j-1])
13              {  echange(&tab[j],&tab[j-1]);
14                 permut = 1;
                }
            }
          }

15        i++;
       }
16     while(permut);
   }
```

Exercice 2

Propriétés du tri cocktail :

- **En place :** oui puisqu'on travaille directement sur le tableau à trier.
- **Stable :** oui car on n'échange des valeurs qu'en cas d'inégalité stricte (donc tout ordre préétabli est préservé).

Exercice 3

Complexité du tri cocktail :

- **taille des données :** taille du tableau N.
- fonction `echange` :
  - **complexité temporelle :**
    - lignes 2,3,4 : opérations élémentaires.
    - **total** : $T_e(N) = O(1)$.
  - **complexité spatiale** :
    - 3 variables de type simple.
    - **total** : $S_e(N) = O(1)$.
- fonction de tri :
  - **complexité temporelle :**
    - ligne 0 : op.élém.
    - ligne 1 & 16 : `do`
      - condition : op. élém.
      - répétitions : $N - 1$ au pire des cas.
      - lignes 2,3,4 : opérations élémentaires.
      - ligne 5 : `if`
        - condition : opération élémentaire.
        - then :
          - ligne 6 : `for`
            - init/cond/incr : op. élém.
            - répétitions : $N - 1$ au pire des cas.
            - ligne 7 : `if`
              - condition : op. élémentaire.
              - ligne 8 : `echange` : $O(1)$.
              - ligne 9 : op.élém.
              - total : $O(1)$.
            - total : $(N - 1) \times O(1) = O(N)$.
        - ligne 10 : `else`
          - ligne 11 : `for`
            - init/cond/incr : op. élém.
            - répétitions : $N - 2$ au pire des cas.
            - ligne 12 : `if`
              - condition : op. élémentaire.
              - ligne 13 : `echange` : $O(1)$.
              - ligne 14 : op.élém.
              - total : $O(1)$.
            - total : $(N - 2) \times O(1) = O(N)$.
        - total : $max\big(O(N), O(N)\big) = O(N)$.
      - ligne 15 : op.élém.
      - total : $T(N) = (N - 1) \times O(N) = O(N^2)$.
  - **complexité spatiale :**
    - 5 variables de type simple.
    - 1 tableau de taille N.
    - fonction échange : $O(1)$.
    - total : $S(N) = O(N)$.

2 Comparaison avec le tri à bulle

Exercice 4

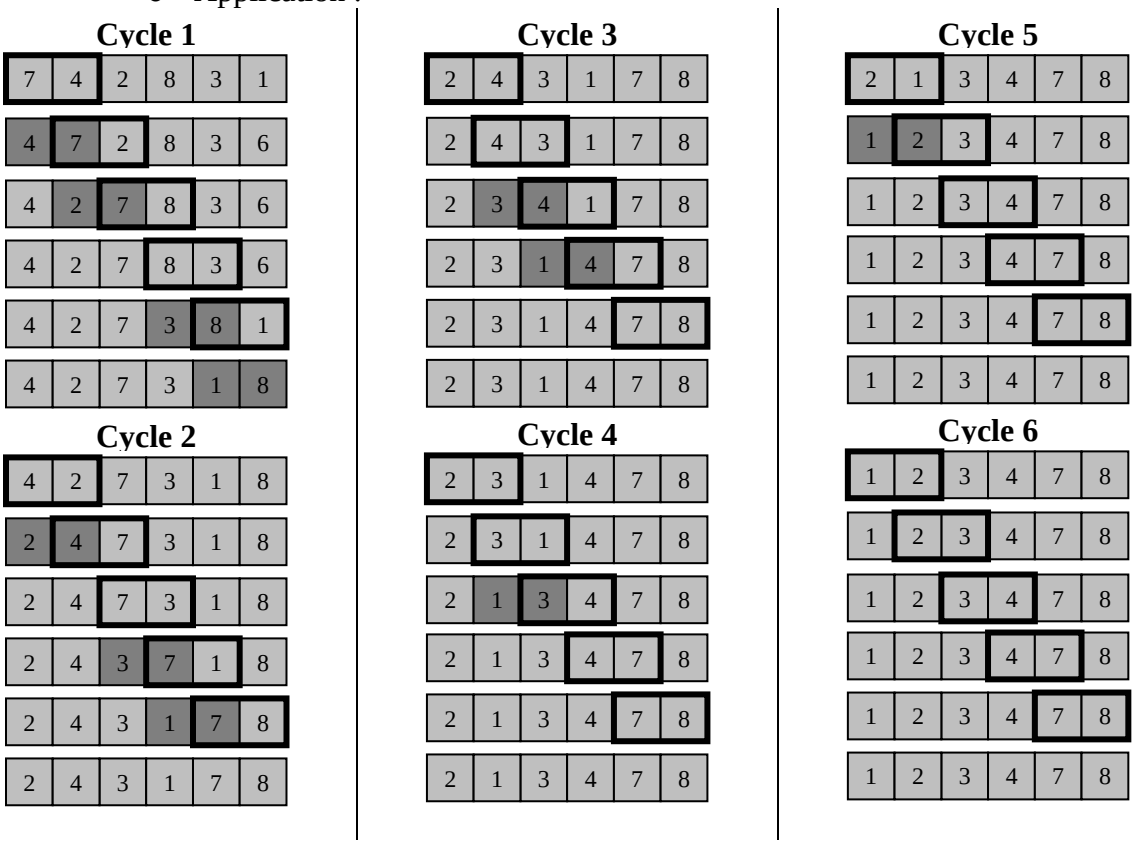
Complexité du tri à bulle (cf. le cours pour le calcul) :

- **complexité spatiale :** : $T(N) = O(N^2)$.
- **complexité temporelle :** $S(N) = O(N)$.

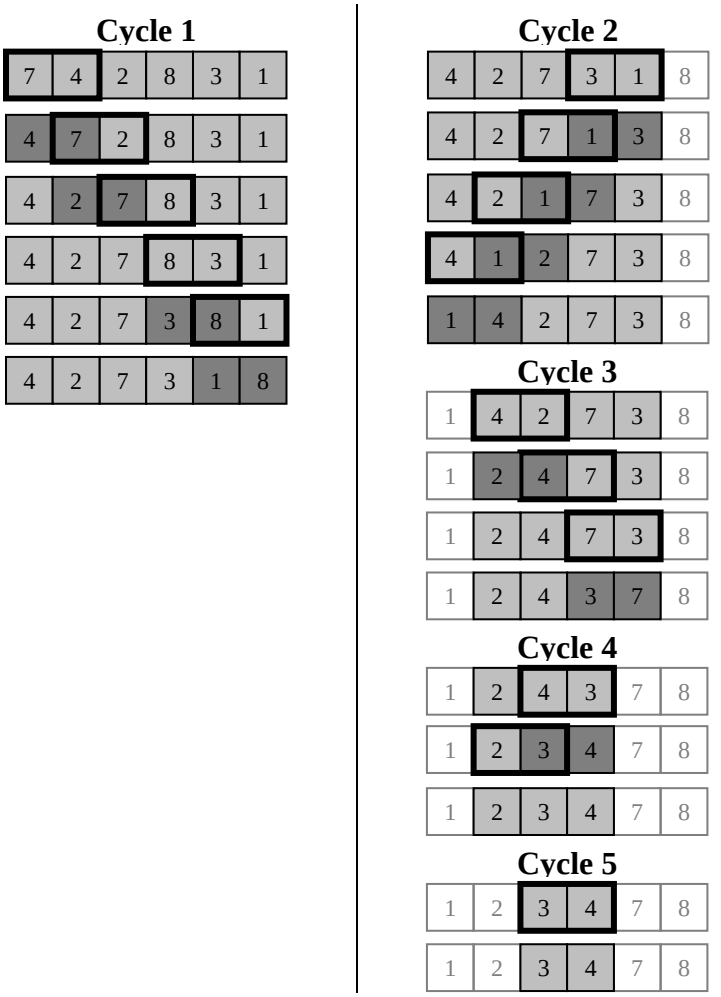
Les deux algorithmes ont les mêmes complexités asymptotiques dans le pire des cas. On ne peut donc pas les départager, puisque cela signifie que pour une valeur de N très grande, ils auront besoin de temps de calcul et d'espaces mémoire du même ordre de grandeur.

Exercice 5

- Tri à bulle :
  - Application :



  - Nombre de comparaisons : $6 \times 5 = 30$.
  - Nombre d'échanges : $4 + 3 + 2 + 1 + 1 + 0 = 11$.
- Tri cocktail :
  - Application :



- Nombre de comparaisons : $5 + 4 + 3 + 2 + 1 = 15$.
- Nombre d'échanges : $4 + 4 + 2 + 1 + 0 = 11$.

- Comparaison :
  - Pour ce tableau, le tri cocktail est plus efficace car il effectue moins de comparaisons (mais autant d'échanges).
  - Or, les complexités asymptotiques montraient que les deux algorithmes étaient temporellement équivalents, donc on ne devrait pas observer de différences ?
  - Mais nous ne nous plaçons pas ici dans des conditions asymptotiques, il n'est donc pas incohérent d'observer une différence entre les deux algorithmes (les données étant de petite taille).

3 Version récursive

Exercice 6

```
   void tri_cocktail_rec(int tab[N], int deb, int fin, int direction)
0  {   int j,permut=0;

1      if(fin>deb)
       {  // allez
2         if(direction)
3         {  for(j=deb;j<fin;j++)
4            {  if(tab[j]>tab[j+1])
5               {  echange(&tab[j],&tab[j+1]);
6                  permut = 1;
                }
            }
```

```
7            if(permut)
8               tri_cocktail_rec(tab,deb,fin-1,!direction);
          }
          // retour
9         else
10        {  for(j=fin;j>deb;j--)
11           {  if(tab[j]<tab[j-1])
12              {  echange(&tab[j],&tab[j-1]);
13                 permut = 1;
                }
             }
14           if(permut)
15              tri_cocktail_rec(tab,deb+1,fin,!direction);
          }
       }
   }
```

Exercice 7

- **Complexité temporelle :**
  - ligne 0 : op. élém.
  - ligne 1 : `if`
    - condition : op. élém.
    - ligne 2 : `if`
      - condition : op. élém.
      - then :
        - ligne 3 : `for`
          - init/cond/incr :op. élém.
          - répétitions : $N-1$ au pire des cas.
          - ligne 4 : `if`
            - condition : op.élém.
            - ligne 5 : `echange` : $O(1)$.
            - ligne 6 : op. élém.
            - total : $O(1)$.
          - total : $(N-1) \times O(1) = O(N)$.
        - ligne 7 : `if`
          - condition : op.élém.
          - ligne 8 : `tri_cocktail_rec` : $T(N-1)$.
          - total : $T(N-1)$
        - total : $T(N-1) + O(N)$.
      - ligne 9 : `else`
        - ligne 10 : `for`
          - init/cond/incr :op. élém.
          - répétitions : $N-2$ au pire des cas.
          - ligne 11 : `if`
            - condition : op.élém.
            - ligne 12 : `echange` : $O(1)$.
            - ligne 13 : op. élém.
            - total : $O(1)$.
          - total : $(N-2) \times O(1) = O(N)$.
        - ligne 14 : `if`
          - condition : op.élém.
          - ligne 15 : `tri_cocktail_rec` : $T(N-1)$.
          - total : $T(N-1)$

- total : $T(N-1)+O(N)$.
- total : $T(N-1)+O(N)$.
- total : $T(N-1)+O(N)$.
- total : $T(N)=T(N-1)+O(N)$.
- On utilise le formulaire donné en cours :
  - La première formule est : $T(N)=T(N-1)+O(N^k)$.
  - En prenant $k=1$, les deux formules sont équivalentes.
  - On en conclut donc que $T(N)=O(N^{k+1})=O(N^2)$.
  - On obtient la même complexité temporelle que pour la version itérative.

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 58 | représentation des tris

1 Représentation des données

Exercice 1

```
void dessine_valeur(int index, int valeur, int mode)
{  Uint32 coul;

   // calcul de la position et des dimensions
   int largeur = FENETRE_LARGEUR/(float)N;
   int h = round(valeur*FENETRE_HAUTEUR/(float)M);
   int x = round(index*largeur);

   // calcul de la couleur
   switch(mode)
   {  case 0:
         coul = C_NOIR;
      break;
      case 1:
         if(index%2==0)
            coul = convertis_rvb(150,150,150);
         else
            coul = convertis_rvb(100,100,100);
         break;
      case 2:
         if(index%2==0)
            coul = convertis_rvb(255,0,0);
         else
            coul = convertis_rvb(190,0,0);
         break;
      case 3:
         if(index%2==0)
            coul = convertis_rvb(0,0,255);
         else
            coul = convertis_rvb(0,0,190);
         break;
   }

   // trace du rectangle
   remplis_rectangle(x,FENETRE_HAUTEUR-h,largeur,FENETRE_HAUTEUR-1,coul);
}
```

Exercice 2

```
void dessine_tableau(int tab[N], int mode)
{  int i;
   for(i=0;i<N;i++)
      dessine_valeur(i,tab[i],mode);
   raffraichis_fenetre();
}
```

Exercice 3

```
void dessine_sous_tableau(int tab[N], int debut, int fin, int mode)
{  int i;
   for(i=debut;i<=fin;i++)
      dessine_valeur(i,tab[i],mode);
   raffraichis_fenetre();
```

```
}
```

Exercice 4

```
void echange_valeurs(int tab[N], int i, int j)
{  int temp;

   // efface les rectangles precedents
   dessine_valeur(i, tab[i], 0);
   dessine_valeur(j, tab[j], 0);
   // met a jour le tableau
   temp = tab[i];
   tab[i] = tab[j];
   tab[j] = temp;
   // dessine les nouveaux rectangles
   dessine_valeur(i, tab[i], 2);
   dessine_valeur(j, tab[j], 2);
   // met a jour l'ecran
   raffraichis_fenetre();
   attends_delai(DELAI);

   // redessine les nouveaux rectangles en gris
   dessine_valeur(i, tab[i], 1);
   dessine_valeur(j, tab[j], 1);
}
```

Exercice 5

```
void ecrase_valeur(int tab[N], int i, int valeur)
{  // efface le rectangle precedent
   dessine_valeur(i, M, 0);
   // met a jour le tableau
   tab[i] = valeur;
   // dessine le nouveau rectangle en rouge
   dessine_valeur(i, tab[i], 2);
   // met a jour l'ecran
   raffraichis_fenetre();
   attends_delai(DELAI);

   // redessine le nouveau rectangle en gris
   dessine_valeur(i, tab[i], 1);
}
```

2 Algorithmes de tri

Exercice 6

```
void tri_bulles(int tab[N])
{  int permut,i,j=0;
   dessine_tableau(tab,1);
   attends_touche();

   do
   {  permut = 0;
      for (i=1;i<N-j;i++)
      {  if (tab[i-1]>tab[i])
         {  echange_valeurs(tab,i,i-1);
            permut = 1;
         }
      }
      dessine_valeur(N-1-j,tab[N-1-j],3);
      j++;
   }
   while(permut);

   dessine_tableau(tab,3);
   raffraichis_fenetre();
}
```

Exercice 7

```
void tri_selection(int tab[N])
{  int m,i,j;
   dessine_tableau(tab,1);
   attends_touche();

   for(i=N-1;i>0;i--)
   {  m=0;
      for (j=0;j<=i;j++)
      {  if(tab[j]>tab[m])
            m = j;
      }
      echange_valeurs(tab,i,m);
      dessine_valeur(i,tab[i],3);
   }

   dessine_valeur(0,tab[0],3);
   raffraichis_fenetre();
}
```

Exercice 8

```
void tri_insertion(int tab[N])
{  int i,j,temp;
   dessine_tableau(tab,1);
   attends_touche();

   for (i=1;i<N;i++)
   {  temp = tab[i];
      j = i - 1;
      while (j>=0 && tab[j]>temp)
      {  //tab[j+1] = tab[j];
         ecrase_valeur(tab,j+1,tab[j]);
         dessine_valeur(j+1,tab[j+1],3);
         j--;
      }
      //tab[j+1] = temp;
      ecrase_valeur(tab,j+1,temp);
      dessine_valeur(j+1,tab[j+1],3);
   }

   raffraichis_fenetre();
}
```

Exercice 9

On n'a besoin que de modifier deux des fonctions du tri fusion :

- Fonction `calcule_fusion` :

```
void calcule_fusion(int tab[N], int tab1[N], int debut1, int fin1, int tab2[N],
                                                          int debut2, int fin2)
{  int i=debut1,j=debut2,k=debut1,m;

   while(i<=fin1 && j<=fin2)
   {  if(tab1[i]<tab2[j])
      {  ecrase_valeur(tab,k,tab1[i]);
         i++;
      }
      else
      {  ecrase_valeur(tab,k,tab2[j]);
         j++;
      }
      k++;
   }
   if(i<=fin1)
   {  for(m=i;m<=fin1;m++)
      {  ecrase_valeur(tab,k,tab1[m]);
         k++;
```

```
        }
    }
    else
    {   for(m=j;m<=fin2;m++)
        {   ecrase_valeur(tab,k,tab2[m]);
            k++;
        }
    }

    dessine_sous_tableau(tab,debut1,fin2,3);
}
```

- Fonction `tri_fusion` :

```
void tri_fusion(int tab[N])
{   dessine_tableau(tab,1);
    attends_touche();

    tri_fusion_rec(tab,0,N-1);

    raffraichis_fenetre();
}
```

Exercice 10

- Fonction `tri_rapide_rec` :

```
void tri_rapide_rec(int tab[N], int d, int f)
{   int taille=f-d+1;
    int i,j;

    if(taille==1)
        dessine_valeur(d,tab[d],3);
    else if(taille>1)
    {   j = d;
        for(i=d+1;i<=f;i++)
        {   if(tab[i]<tab[d])
            {   j++;
                echange_valeurs(tab,i,j);
            }
        }
        echange_valeurs(tab,d,j);
        dessine_valeur(j,tab[j],3);

        if(j>0)
            tri_rapide_rec(tab,d,j-1);
        if(j<N)
            tri_rapide_rec(tab,j+1,f);
    }
}
```

- Fonction `tri_rapide` :

```
void tri_rapide(int tab[N])
{   dessine_tableau(tab,1);
    attends_touche();

    tri_rapide_rec(tab,0,N-1);

    raffraichis_fenetre();
}
```

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 59 | tri de listes

1 Tri par sélection

Exercice 1

```
element_d* identifie_extremum(liste_d l, int mode)
{  element_d* result = l.debut;
   element_d* e = l.debut;

   while(e!=NULL )
   {  if((!mode && e->valeur < result->valeur)
         || (mode && e->valeur > result->valeur))
         result = e;
      else
         e = e->suivant;
   }

   return result;
}
```

Exercice 2

```
void tri_selection_rec(liste_d *l)
{  element_d *e;
   if(l->debut != l->fin)
   {  e = identifie_extremum(*l,0);
      detache_element(l,e);
      tri_selection_rec(l);
      insere_element_d(l,e,0);
   }
}
```

Exercice 3

```
void tri_selection_it(liste_d *l)
{  element_d *e;
   liste_d temp;

   temp.debut = NULL;
   temp.fin = NULL;
   while(l->debut!=NULL)
   {  e = identifie_extremum(*l,1);
      detache_element(l,e);
      insere_element_d(&temp,e,0);
   }
   l->debut = temp.debut;
   l->fin = temp.fin;
}
```

2 Tri par insertion

Exercice 4

```
void insere_element_trie(liste_d *l, element_d *e)
{  element_d *temp = l->debut, *avant, *apres;

   if(temp==NULL || temp->valeur>e->valeur)
```

```
      {  if(temp != NULL)
            temp->precedent = e;
         else
            l->fin = e;
         e->suivant = temp;
         l->debut = e;
      }
      else
      {  do
         {  avant = temp;
            temp = temp->suivant;
         }
         while(temp!=NULL && temp->valeur<e->valeur);
         apres = temp;
         if(apres != NULL)
            apres->precedent = e;
         else
            l->fin = e;
         e->suivant = apres;
         e->precedent = avant;
         avant->suivant = e;
      }
   }
```

Exercice 5

```
   void trie_insertion(liste_d *l)
   {  element_d *temp=l->debut;
      liste_d triee, non_triee;

      if(temp!=NULL)
      {  // initialisation des deux listes
         triee.debut = NULL;
         triee.fin = NULL;
         non_triee.debut = l->debut;
         non_triee.fin = l->fin;

         // traitement iteratif
         while(non_triee.debut!=NULL)
         {  temp = non_triee.debut;
            detache_element(&non_triee,temp);
            insere_element_trie(&triee,temp);
         }
         l->debut = triee.debut;
      }
   }
```

3 Tri fusion

Exercice 6

```
   element_d* renvoie_milieu(liste_d l)
   {  element_d *e = l.debut;
      int longueur = 0;

      while(e!=NULL)
      {  longueur++;
         e = e->suivant;
      }
      if(longueur>0)
         e = accede_element_d(l,longueur/2);
      return e;
   }
```

Exercice 7

```
   void calcule_division(liste_d l, liste_d *lg, liste_d *ld)
   {  element_d *deb_d = renvoie_milieu(l);
```

```
    element_d *fin_g = deb_d->precedent;

    lg->debut = l.debut;
    lg->fin = fin_g;
    ld->debut = deb_d;
    ld->fin = l.fin;

    fin_g->suivant = NULL;
    deb_d->precedent = NULL;
}
```

Exercice 8

```
void calcule_fusion(liste_d *l, liste_d lg, liste_d ld)
{  element_d *e=NULL,*temp;
   element_d *eg=lg.debut,*ed=ld.debut;
   int vg, vd;

   while(eg!=NULL && ed!=NULL)
   {  vg = eg->valeur;
      vd = ed->valeur;
      if(vg<vd)
      {  temp = eg;
         eg = eg->suivant;
      }
      else
      {  temp = ed;
         ed = ed->suivant;
      }
         if(e==NULL)
      {  e = temp;
         l->debut = temp;
      }
      else
      {  temp->precedent = e;
         e->suivant = temp;
         e = temp;
      }
   }

   if(eg!=NULL)
   {  eg->precedent = e;
      e->suivant = eg;
      l->fin = lg.fin;
   }
   else
   {  ed->precedent = e;
      e->suivant = ed;
      l->fin = ld.fin;
   }
}
```

Exercice 9

```
void tri_fusion(liste_d *l)
{   liste_d lg,ld;
   lg.debut = NULL;
   lg.fin = NULL;
   ld.debut = NULL;
   ld.fin = NULL;

   if(l->debut!=NULL && l->debut->suivant!=NULL)
   {  calcule_division(*l,&lg,&ld);
      tri_fusion(&lg);
      tri_fusion(&ld);
      calcule_fusion(l,lg,ld);
   }
}
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 60 | représentation de l'ADN

1 Codage de l'information

Exercice 2

```
int est_adn(liste_s brin)
{  int resultat = 1;
   element_s *e = brin.debut;
   liste_s l;

   if(e != NULL)
   {  if(e->valeur=='A' || e->valeur=='C' || e->valeur=='G' || e->valeur=='T')
      {  l.debut = e->suivant;
         resultat = est_adn(l);
      }
      else
         resultat = 0;
   }
   return resultat;
}
```

Exercice 3

```
int genere_adn(liste_s *brin, int n)
{  int i=0, base_index,erreur=0;
   char base;
   element_s *e;
   while(i<n && !erreur)
   {  base_index = rand()/(((float)RAND_MAX+1)/(4-1+1))+1;
      switch(base_index)
      {  case 1:
            base = 'A';
            break;
         case 2:
            base = 'C';
            break;
         case 3:
            base = 'G';
            break;
         case 4:
            base = 'T';
            break;
      }
      if((e = cree_element_s(base)) == NULL)
         erreur = -1;
      else
      {  if((erreur=insere_element_s(brin,e,0)) != -1)
            i++;
      }
   }
   return erreur;
}
```

2 Structure en double hélice

Exercice 4

```
int calcule_complementaire(liste_s original, liste_s *complementaire)
{  element_s *e = original.debut;
   char base_orig,base_comp;
   liste_s l2;
   int erreur = 0;

   if(e != NULL)
   {  // on traite d'abord le reste de la liste
      l2.debut = e->suivant;
      if((erreur=calcule_complementaire(l2,complementaire)) != -1)
      {  // on determine la base complementaire
         base_orig = e->valeur;
         switch(base_orig)
         {  case 'A':
            base_comp = 'T';
               break;
            case 'C':
               base_comp = 'G';
               break;
            case 'G':
               base_comp = 'C';
               break;
            case 'T':
               base_comp = 'A';
               break;
         }
         // on cree l'element pour le brin complementaire
         if((e = cree_element_s(base_comp)) == NULL)
            erreur = -1;
         else
            erreur = insere_element_s(complementaire,e,0);
      }
   }
   return erreur;
}
```

Exercice 5

```
int sont_egales(liste_s brin1, liste_s brin2)
{  element_s *e1=brin1.debut,*e2=brin2.debut;
   liste_s l1,l2;
   int result;

   if(e1 == NULL)
      result = (e2 == NULL);
   else if(e2 == NULL)
      result = 0;
   else
   {  if(e1->valeur == e2->valeur)
      {  l1.debut = e1->suivant;
         l2.debut = e2->suivant;
            result = sont_egales(l1,l2);
      }
      else
         result = 0;
   }
   return result;
}
```

Exercice 6

```
int sont_complementaires(liste_s brin1, liste_s brin2)
{  int resultat = -1;
   liste_s comp;
   comp.debut = NULL;

   if(calcule_complementaire(brin1,&comp) != -1)
      resultat = sont_egales(brin2,comp);
   return resultat;
```

```
}
```

Exercice 7

- Taille des données : longueur N de la séquence constituant le brin.
- Fonction `sont_complementaires` :
  - lignes 1,2 : opérations élémentaires.
  - ligne 3 : `if`
    - condition :
      - appel de la fonction `calcule_complementaire` : $_{cc}(N)$.
      - comparaison : opération élémentaire.
    - ligne 4 :
      - affectation : opération élémentaire.
      - appel de la fonction `egale` : $T_e(N)$.
    - total : $_{cc}(N) + \ _e(N) + O(1)$.
  - ligne 5 : opération élémentaire.
  - total : $T(N) = \ _{cc}(N) + \ _e(N) + O(1)$.
- Fonction `calcule_complementaire` :
  - lignes 1,2 : opérations élémentaires.
  - ligne 3 : `if`
    - condition : opération élémentaire.
    - ligne 4 : opération élémentaire.
    - ligne 5 : `if`
      - condition :
        - affectation/comparaison : opérations élémentaires.
        - appel de `calcule_complementaire` : $_{cc}(N-1)$.
      - ligne 6 : opération élémentaire.
      - ligne 7 : `switch`
        - lignes 8-19 : opérations élémentaires.
        - total : max des case, i.e. : $O(1)$.
      - ligne 20 : `if`
        - condition :
          - affectation/comparaison : op.élémentaires.
          - appel de `cree_element_s` : $O(1)$.
        - then :
          - ligne 21 : opération élémentaire
        - ligne 22 : `else`
          - ligne 23 :
            - affectation : opération élémentaire.
            - appel d'`insere_element_s` en première position : $O(1)$.
        - total : $O(1)$.
      - total : $_{cc}(N-1) + O(1)$.
    - total : $_{cc}(N-1) + O(1)$.
  - total : $_{cc}(N) = \ _{cc}(N-1) + O(1)$.
  - on applique le formulaire du cours :
    - première formule : $T(N) = T(N-1) + O(N^k)$.
    - ici, on a $k = 0$, d'où la complexité $O(N^{1+1})$.
    - $_{cc}(N) = O(N)$.
- Fonction `sont_egales` :

- ligne 1 : opérations élémentaires.
- ligne 2 : `if`
  - condition : opération élémentaire.
  - then :
    - ligne 3 : opérations .élémentaires.
  - ligne 4 : `else`
    - ligne 4 : `if`
      - condition : opération élémentaire.
      - then :
        - ligne 5 : opération élémentaire.
      - ligne 6 : `else`
        - ligne 7 : `if`
          - condition : opération élémentaire.
          - then :
            - lignes 8,9 : op.élémentaires.
            - ligne 10 : appel de `egale` : $_e(N-1)$.
          - ligne 11 : `else`
            - ligne 12 : op.élémentaire.
          - total : $T_e(N-1)+O(1)$.
      - total : $T_e(N-1)+O(1)$.
  - total : $T_e(N-1)+O(1)$.
- ligne 13 : opération élémentaire.
- total : $T_e(N) = \ _e(N-1)+O(1)$.
- on applique le formulaire du cours :
  - même formule que pour `calcule_complementaire`.
  - $_e(N)=O(N)$.

- Complexité de `sont_complementaires` :
  - $(N) = \ _{cc}(N) + \ _e(N)+O(1)=O(N)+O(N)+O(1)$.
  - $(N)=O(N)$.

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 61 | nombres de grande taille

1 Représentation

Exercice 1

```
int verifie_gn(liste_s l)
{  int result = 0;
   element_s *e=l.debut;
   liste_s temp;
   if(e==NULL)
      result = 1;
   else if(e->valeur>=0 && e->valeur<=9)
   {  temp.debut = e->suivant;
      result = verifie_gn(temp);
   }
   return result;
}
```

Exercice 2

La récursivité est simple, puisqu'au cours d'une l'exécution de la fonction, elle s'appelle seulement elle-même (donc ce n'est pas une récursivité croisée), et au maximum une seule fois (donc ce n'est pas une récursivité multiple).

```
   int compte_chiffres(liste_s l)
1  {   int resultat = 0;                          O(1)
       Liste_s temp;
2      if(l.debut != NULL)                        O(1)
3      {  temp.debut = l.debut->suivant;          O(1)
4         resultat = 1 + compte_chiffres(temp);   O(1) + T(N − 1)
       }
5      return resultat;                           O(1)
   }
```

Taille des données :

- La taille de la liste, notée N.

Complexité temporelle :

- Complexité du **cas d'arrêt** $= 0$ (liste vide) :
  - ligne `1` : affectation : $O(1)$
  - ligne `2` : condition du `if` : $O(1)$
  - ligne `5` : exécution de `return` : $O(1)$
  - **total :** $T(0) = O(1) + O(1) + O(1) = O(1)$.
- Complexité du **cas général** > 0 :
  - ligne `1` : affectation : $O(1)$
  - ligne `2` : dans le `if` :
    - condition : $O(1)$
    - ligne `3` : affectation : $O(1)$
    - ligne `4` : affectation, opération élémentaire et appel récursif sur une liste plus courte de un élément $O(1) + T(N-1)$
    - **total** du `if` **:** $O(1) + T(N-1)$

- ligne `5` : exécution de `return` : $O(1)$
- **total :** $T(N) = O(1) + T(N-1)$

- On utilise le **formulaire** :
  - la première formule dit que si $T(n) = T(n-1) + \Theta(n^k)$, alors la complexité de l'algorithme est en $O(n^{k+1})$.
  - ici on a $k = 0$, donc la complexité temporelle est en $O(n^{0+1}) = O(n)$

2 Affichage et conversion

Exercice 3

```
void affiche_gn_it(liste_s l)
{  int i,n = compte_chiffres(l);
   element_s *e;
   for(i=n-1;i>=0;i--)
   {  e = accede_element_s(l, i);
      printf("%d",e->valeur);
      if(i!=0 && i%3==0)
         printf(" ");
   }
}
```

Exercice 4

```
void affiche_gn_rec(liste_s l, int cpt)
{  element_s *e=l.debut;
   liste_s temp;
   if(e != NULL)
   {  temp.debut = e->suivant;
      // on affiche d'abord les chiffres de rang plus élevé
      affiche_gn_rec(temp,cpt+1);
      // on insère éventuellement un espace
      if(cpt>0 && cpt%3==0 && e->suivant!=NULL)
         printf(" ");
      // on affiche le chiffre courant
      printf("%d",l.debut->valeur);
   }
}
```

Exercice 5

```
int convertis_chaine(char *chaine, liste_s *l)
{  char *p = chaine;
   element_s *e;
   int erreur = 0;

   while(*p!='\0' && erreur==0)
   {  if((e=cree_element_s(*p - '0')) == NULL)
         erreur = -1;
      else
      {  erreur = insere_element_s(l,e,0);
         p++;
      }
   }
   return erreur;
}
```

3 Opérations

Exercice 6

```
int additionne_gn(liste_s l1, liste_s l2, liste_s *res)
{  int erreur,retenue=0,valeur,cpt=0;
   element_s *e1=l1.debut,*e2=l2.debut,*er;

   while(erreur!=-1 && e1!=NULL)
```

```
      {  // on calcule la valeur de l'element
         valeur = e1->valeur + retenue;
         if(e2!=NULL)
            valeur = valeur + e2->valeur;
         retenue = valeur / 10;
         valeur = valeur % 10;

         // on rajoute un element dans la liste
         if((er = cree_element_s(valeur)) == NULL)
            erreur = -1;
         else
            insere_element_s(res,er,cpt);

         // on passe aux chiffres suivants
         if(e1!=NULL)
            e1 = e1->suivant;
         else
         {  e1 = e2;
            e2 = NULL;
         }
         if(e2!=NULL)
            e2 = e2->suivant;
         cpt++;
      }

      return erreur;
   }
```

Exercice 7

```
   int multiplie_gn_p10(liste_s *l, int p)
1  {   int erreur=0;                                          O(1)
2      element_s *e;                                          O(1)
3      if(p>0)                                                O(1)
4      {  if((e=cree_element_s(0)) == NULL)                   O(1) + O(1)
5            erreur = -1;                                     O(1)
6         else
7            erreur = insere_element_s(l,e,0);                O(1) + O(1)
8         if(erreur != -1)                                    O(1)
9            multiplie_gn_p10(l,p-1);                         O(1) + T(p − 1)
       }
10     return erreur;                                         O(1)
   }
```

Taille des données :

- La taille des données correspond ici non seulement à la longueur *n* de la liste, mais aussi à `p`. Comme on a simplifié la complexité de la fonction `insere_element` à $O(1)$, on peut ignorer *n* et n'utiliser que `p`.
- **remarque :** l'insertion d'un élément en début de liste se fait en temps constant, d'où la simplification effectuée dans le sujet. Mais en réalité, la complexité dans le pire des cas de cette fonction est en $O(n)$.

Complexité temporelle :

- Complexité du **cas d'arrêt** $p = 0$ (exposant nul, indépendamment de la longueur de la liste) :
  - ligne `1` et `2` : affectation : $O(1)$
  - ligne `3` : condition du `if` : $O(1)$
  - ligne `10` : exécution de `return` : $O(1)$
  - **total :** $T(n, 0) = O(1)$.
- Complexité du **cas général** $n > 0$:
  - ligne `1` et `2` : affectation : $O(1)$

- ligne `3` : dans le `if`
  - condition : $O(1)$
  - ligne `4` : dans le `if` :
    - condition : affectation et test : $O(1)$
    - premier bloc :
      - ligne `5` : affectation : $O(1)$
    - ligne `6` : bloc `else`
      - ligne `7` : affectation et appel de fonction : $O(1) + O(1)$
    - **total** du `if` : $\max\big(O(1), O(1)\big) = O(1)$
  - ligne `8` : dans le `if`
    - condition : $O(1)$
    - ligne `9` : appel récursif $T(p-1)$
    - **total** du `if` : $O(1) + T(p-1)$
  - **total** du `if` : $\max\big(O(1), T(p-1)\big) = T(p-1)$
- ligne `10` : exécution de `return` : $O(1)$
- **total** de la fonction : $T(p) = O(1) + T(p-1)$
- la relation est de la même forme que lors du calcul de complexité précédent, et on a donc $T(p) = O(p)$.

Exercice 8

```
int multiplie_gn_chiffre(liste_s l, int c, liste_s *res)
{  int erreur=0, retenue=0, unite, produit, cpte=0;
   element_s *e = l.debut, *temp;

   // calcul principal
   while(e!=NULL && erreur!=-1)
   {  produit = c*e->valeur + retenue;
      unite = produit % 10;
      retenue = produit / 10;
      if((temp=cree_element_s(unite)) == NULL)
         erreur = -1;
      else
      {  erreur = insere_element_s(res,temp,cpte);
         if(erreur != -1)
         {  e = e->suivant;
            cpte++;
         }
      }
   }

   // retenue de débordement
   if(retenue>0 && erreur!=-1)
   {  if((temp=cree_element_s(retenue)) == NULL)
         erreur = -1;
      else
         erreur = insere_element_s(res,temp,cpte);
   }

   return erreur;
}
```

Exercice 9

```
int multiplie_gn(liste_s l1, liste_s l2, liste_s* res)
{  int erreur=0,cpte=0;
   liste_s temp1,temp2;
   element_s *e2 = l2.debut;

   // init
```

```
      temp1.debut = NULL;
      temp2.debut = NULL;

      while(erreur!=-1 && e2!=NULL)
      {  // multiplication de l1 par un chiffre de l2
         erreur = vide_liste_s(&temp1);
         if(erreur!=-1)
            erreur = multiplie_gn_chiffre(l1,e2->valeur,&temp1);

         // on décale le résultat de façon appropriée
         if(erreur != -1)
            erreur = multiplie_gn_p10(&temp1,cpte);

         // on ajoute le résultat à la somme courante
         if(erreur != -1)
            erreur = vide_liste_s(&temp2);
         if(erreur != -1)
         {  temp2.debut = res->debut;
            res->debut = NULL;
            erreur = additionne_gn(temp1,temp2,res);
         }

         // on passe au chiffre suivant
         if(erreur != -1)
         {  e2 = e2->suivant;
            cpte++;
         }
      }

      return erreur;
   }
```

4 Tri

Exercice 10

```
   int compare_gn(liste_s l1, liste_s l2)
   {   int result;
1      element_s *e1=l1.debut,*e2=l2.debut;
       liste_s temp1,temp2;

2      if(e1==NULL)
3      {  if(e2==NULL)
4             result = 0;
5          else
6             result = -1;
       }
7      else
8      {  if(e2==NULL)
9             result = +1;
10         else
11         {  temp1.debut = e1->suivant;
12            temp2.debut = e2->suivant;
13            result = compare_gn(temp1,temp2);
14            if(result==0)
15               result = e1->valeur - e2->valeur;
           }
       }

16     return result;
   }
```

Taille des données :

- Longueur des listes comparées, que l'on notera M_1 et M_2.

Complexité temporelle :

- **Cas simple** : au moins l'une des deux listes est vide, i.e. $M_1 = 0$ ou $M_2 = 0$.

    - ligne `1` : affectation : $O(1)$.
    - ligne `2` : `if`
        - condition : $O(1)$.
        - ligne `3` : `if`
            - condition : $O(1)$.
            - ligne `4` : premier bloc : affectation : $O(1)$.
            - ligne `5` : bloc `else` : affectation : $O(1)$.
            - total : $O(1) + \max\big(O(1), O(1)\big) = O(1)$.
    - ligne `16` : `return` : $O(1)$.
    - total : $T(M_1, 0) = T(0, M_2) = T(0,0) = O(1)$.
- **Cas général** : les deux listes sont non-vides, i.e. $M_1 \neq 0$ et $M_2 \neq 0$.
    - ligne `1` : affectation : $O(1)$.
    - ligne `2` : `if`
        - condition : $O(1)$.
        - ligne `7` : bloc `else`.
            - ligne `8` : `if`.
                - condition : $O(1)$.
                - ligne `9` : premier bloc : affectation : $O(1)$.
                - ligne `10` : bloc `else`.
                    - ligne `11` : affectation : $O(1)$.
                    - ligne `12` : affectation : $O(1)$.
                    - ligne `13` : appel récursif sur des listes plus petites de un élément, soit : $T(M_1 - 1, M_2 - 1)$.
                        - ligne `14` : `if`.
                            - ligne `15` : premier bloc : affectation : $O(1)$.
                        - total du `if` : $O(1)$.
                    - total du bloc `else` : $O(1) + T(M_1 - 1, M_2 - 1)$.
                - total du `if` : $O(1) + \max\big(O(1), O(1) + T(M_1 - 1, M_2 - 1)\big) = O(1) + T(M_1 - 1, M_2 - 1)$.
            - total du bloc `else` : pareil, $O(1) + T(M_1 - 1, M_2 - 1)$.
        - total du `if` : $O(1) + T(M_1 - 1, M_2 - 1)$
    - total : $T(M_1, M_2) = O(1) + T(M_1 - 1, M_2 - 1)$.
- Simplification :
    - La fonction s'arrête dès qu'on arrive à la fin d'une des deux listes, donc on peut ne considérer qu'une seule des deux longueurs : la plus courte $M = \min(M_1, M_2)$.
    - Cela permet de réécrire la complexité :
        - $T(0) = O(1)$.
        - $T(M) = O(1) + T(M - 1)$
    - On peut ainsi appliquer le formulaire pour résoudre la relation de récurrence, et obtenir une complexité $T(M) = O(M)$.

Exercice 11

La fonction donnée dans le sujet implémente le tri par insertion, qui consiste à diviser le tableau en une partie triée et une partie non-triée, et à insérer itérativement tous les éléments non-triés à la bonne position dans la partie triée. Sa complexité temporelle est en $O(N^2)$, où N est la taille du tableau.

Pour adapter la fonction au tri de grands nombres, il suffit de modifier le type du tableau `tab` et de la variable `temp`, et d'utiliser la fonction `compare_nombres` à la place de l'opérateur `>` dans la condition du `while`.

```
   void trie_gn(liste_s tab[N])
   {   int i,j;
       Liste_s temp;
1      for(i=1;i<N;i++)                                  ..... O(1)
2      {  temp = tab[i];                                 ..... O(1)
3         j = i-1;                                       ..... O(1)
4         while(j>=0 && compare_gn(tab[j],temp)>0)       ..... O(M)
5         {  tab[j+1] = tab[j];                          ..... O(1)
6            j--;                                        ..... O(1)
          }
7         tab[j+1] = temp;                               ..... O(1)
       }
   }
```

O(MN) : lignes 4–6 ; O(MN²) : lignes 1–7

Taille des données :

- Taille des listes représentant les nombres à trier : M.
- Taille du tableau : N.

Complexité temporelle :

- ligne `1` : `for` :
  - initialisation, condition, affectation : $O(1)$.
  - lignes `2` et `3` : opérations élémentaires : $O(1)$.
  - ligne `4` : `while` :
    - condition : affectation et `compare_nombres` : $O(1) + O(M)$
    - lignes `5` et `6` : opérations élémentaires : $O(1)$.
    - répétitions : $N - 1$ au pire.
    - **total :** $(N - 1) \times O(M) = O(MN)$.
  - ligne `7` : affectation : $O(1)$.
  - répétitions : $N - 1$ au pire.
  - **total :** $(N - 1) \times O(MN) = O(MN^2)$.
- **total :** $T(M, N) = O(MN^2)$.

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 62 | table de symboles

1 Structure de données

Exercice 1

- Type modifié dans le fichier `.h` :

```
typedef struct s_element_s
{  char* cle;
   int valeur;
   struct s_element_s *suivant;
} element_s;
```

- Fonction de création d'élément modifiée dans le fichier `.c` :

```
element_s* cree_element_s(char* cle, int valeur)
{  element_s *e;
   if((e = (element_s *) malloc(sizeof(element_s))) != NULL)
   {  e->cle = cle;
      e->valeur = valeur;
      e->suivant = NULL;
   }
   return e;
}
```

- Fonction d'affichage de la liste modifiée dans le fichier `.c` :

```
void affiche_liste_s(liste_s l)
{  element_s *temp = l.debut;

   printf("{");
   while(temp != NULL)
   {  printf(" %s:%d", temp->cle, temp->valeur);
      temp = temp->suivant;
   }
   printf(" }\n");
}
```

Exercice 2

- Type demandé :

```
typedef liste_s tabsymb;
```

- Fonction de création :

```
tabsymb cree_tabsymb()
{  liste_s l;
   l.debut = NULL;
   return l;
}
```

Exercice 3

```
int ajoute_symb(tabsymb *t, char *cle)
{  int erreur = 0;
   element_s *e = t->debut;
   char *temp;

   // on cherche l'element
   while(e!=NULL && compare_chaines(e->cle,cle)!=0)
      e = e->suivant;
   // s'il n'existe pas : on le cree
```

```
      if(e==NULL)
      {  if((temp=copie_chaine_dyn(cle))==NULL)
            erreur = -1;
         else
         {  if((e=cree_element_s(temp,1))==NULL)
               erreur = -1;
            else
               erreur = insere_element_s(t,e,0);
         }
      }
      // sinon on se contente de l'incrementer
      else
         e->valeur++;

      return erreur;
   }
```

Exercice 4

```
   int verifie_symb(tabsymb t, char *cle)
   {  int result = 0;
      element_s *e = t.debut;

      while(e!=NULL && compare_chaines(e->cle,cle)!=0)
         e = e->suivant;
      if(e!=NULL)
         result = e->valeur;

      return result;
   }
```

2 Analyse du texte

Exercice 5

```
   int analyse_texte(char *nom_fichier, tabsymb *t)
   {  int erreur = 0;
      FILE *fp;
      char temp[255];

       // on ouvre le fichier
      if((fp = fopen(nom_fichier,"r"))==NULL)
         erreur = -1;
      else
       {  // on traite chaque mot
         while(fscanf(fp,"%s ",temp)!=EOF && erreur!=-1)
            erreur = ajoute_symb(t,temp);
         // on verifie qu'il n'y a pas eu d'erreur
         if(!feof(fp))
            erreur = -1;
       }

       // on ferme le fichier
       if((fclose(fp))==EOF)
         erreur = -1;

      return erreur;
   }
```

3 Tri des symboles

Exercice 6

```
   int compare_elements(element_s *e1, element_s *e2)
   {  int resultat = e1->valeur - e2->valeur;
      if(resultat==0)
         resultat = compare_chaines(e1->cle,e2->cle);
```

```
   return resultat;
}
```

Exercice 7

```
element_s* etete_liste(liste_s *l)
{  element_s* resultat = l->debut;
   l->debut = resultat->suivant;
   resultat->suivant = NULL;
   return resultat;
}
```

Exercice 8

```
void insere_element_dec(liste_s *l, element_s *e)
{  element_s *e1,*e2=l->debut;

   // insertion en debut de liste
   if(compare_elements(e2,e)<0)
   {  e->suivant = e2;
      l->debut = e;
   }

   // insertion a l'interieur de la liste
   else
   {  e1 = e2;
      e2 = e2->suivant;
      while(e2!=NULL && compare_elements(e2,e)>0)
      {  e1 = e2;
         e2 = e2->suivant;
      }
      e->suivant = e2;
      e1->suivant = e;
   }
}
```

Exercice 9

```
void trie_liste_dec(liste_s *l)
{  liste_s l2;
   element_s *e;

   // initialisation
   e = etete_liste(l);
   l2.debut = l->debut;
   l->debut = e;

   // traitement iteratif
   while(l2.debut!=NULL)
   {  e = etete_liste(&l2);
      insere_element_dec(l,e);
   }
}
```

4 Filtrage des symboles

Exercice 10

```
int filtre_mots_vides(char *nom_fichier, tabsymb *t)
{  int erreur=0,trouve,i=0;
   FILE *fp;
   char mot[255];
   element_s *e;

    // on ouvre le fichier
   if((fp = fopen(nom_fichier,"r"))==NULL)
      erreur = -1;
   else
    {  // on traite chaque mot
```

```
          while(fscanf(fp,"%s\n",mot)!=EOF && erreur!=-1)
          {  trouve = 0;
             i = 0;
             e = t->debut;
             while(!trouve && e!=NULL)
             {  if(compare_chaines(mot,e->cle)==0)
                {  supprime_element_s(t,i);
                   trouve = 1;
                }
                else
                {  i++;
                   e = e->suivant;
                }
             }
          }
          // on verifie qu'il n'y a pas eu d'erreur
          if(!feof(fp))
             erreur = -1;
       }

       // on ferme le fichier
       if((fclose(fp))==EOF)
          erreur = -1;

      return erreur;
   }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 63 | plus longue sous-séquence commune

2 Longueur d'une PLSC

Exercice 1

Soient $X = (x_0, \dots, x_{p-1})$ et $Y = (y_0, \dots, y_{q-1})$, $X_i = (x_0, \dots, x_{i-1})$ et $Y_j = (x_0, \dots, x_{j-1})$.

Propriété :

- Pour $1 \leq i \leq p$ et $1 \leq j \leq q$: $l_{i,j} = \begin{cases} 1 + l_{i-1,j-1} \text{ si } x_{i-1} = y_{i-1} \\ \max(l_{i-1,j}, l_{i,j-1}) \text{ sinon} \end{cases}$

Cas de base :

- Montrons que la propriété est vraie pour $i = j = 1$:
  - si $x_0 = y_0$: $l_{1,1} = 1$.
  - si $x_0 \neq y_0$: $l_{1,1} = 0 = \max(l_{0,1}, l_{1,0})$, car par définition $l_{0,1} = l_{1,0} = 0$.
- Donc la propriété est vraie pour $l_{1,1}$.

Cas général :

- On suppose que la propriété est vraie pour les rangs inférieurs, et on veut la prouver pour $l_{i,j}$:
  - si $x_{i-1} = y_{j-1}$ alors la concaténation de x_{i-1} à une $PLSC(X_{i-1}, Y_{j-1})$ est une $PLSC(X_i, Y_j)$. Comme la taille d'une $PLSC(X_{i-1}, Y_{j-1})$ est $l_{i-1,j-1}$ et qu'on concatène un seul élément, alors la taille de $PLSC(X_i, Y_j)$ est $l_{i,j} = l_{i-1,j-1} + 1$.
  - si $x_{i-1} \neq y_{j-1}$, alors toute sous-séquence commune à X_i et Y_j est :
    - Soit commune à X_i et Y_{j-1} si elle ne se termine pas par y_{j-1}.
    - Soit commune à X_{i-1} et Y_j si elle ne se termine pas par x_{i-1}.
    - Donc une $PLSC(X_i, Y_j)$ sera la plus longue séquence entre $PLSC(X_i, Y_{j-1})$ et $PLSC(X_{i-1}, Y_j)$.
    - On en déduit que $l_{i,j} = \max(l_{i-1,j}, l_{i,j-1})$.

Conclusion :

- La relation de récurrence est prouvée.

3 Approche naïve

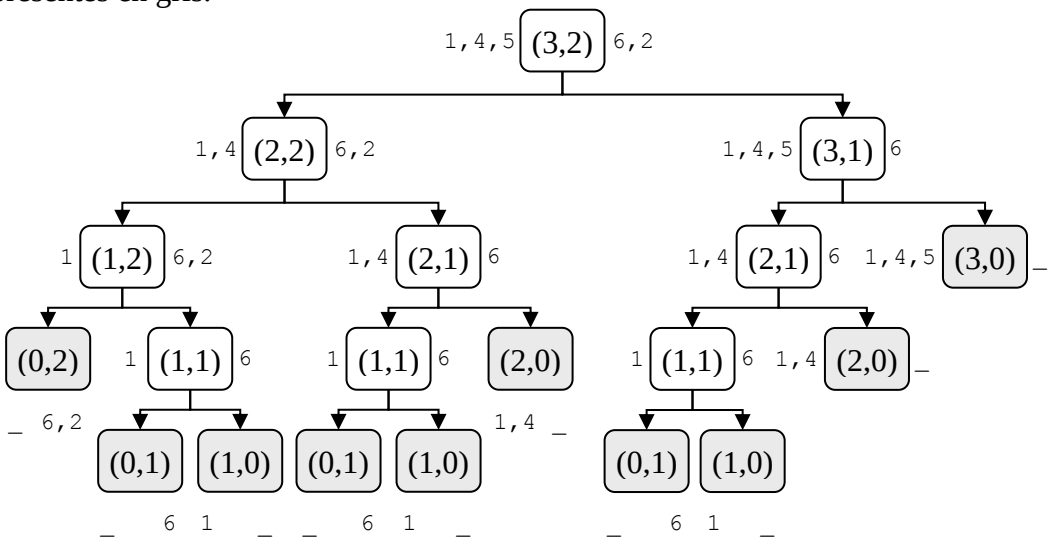
Exercice 2

```
   int plsc1(int tab1[], int taille1, int tab2[], int taille2)
   {   int resultat=0,resultat1,resultat2;

1      if(taille1==0 || taille2==0)
2         resultat = 0;
       else
3      {  if(tab1[taille1-1] == tab2[taille2-1])
4            resultat = 1 + plsc1(tab1, taille1-1, tab2, taille2-1);
          else
5         {  resultat1 = plsc1(tab1, taille1-1, tab2, taille2);
```

```
6            resultat2 = plsc1(tab1, taille1, tab2, taille2-1);
7            if(resultat1>resultat2)
8               resultat = resultat1;
             else
9               resultat = resultat2;
          }
       }
10     return resultat;
   }
```

Exercice 3

Les valeurs indiquées dans les nœuds correspondent aux longueurs des séquences traitées, les séquences elles-mêmes sont indiquées à l'extérieur des nœuds, et les appels terminaux sont représentés en gris.



Exercice 4

Complexité temporelle :

- ligne `1` : `if`
  - ligne `2` : opération élémentaire : $O(1)$.
  - ligne `3` : `if`
    - `then`
      - ligne `4` : appel récursif : $O(1) + T(p-1, q-1)$.
    - `else`
      - ligne `5` : appel récursif : $O(1) + T(p-1, q)$.
      - ligne `6` : appel récursif : $O(1) + T(p, q-1)$.
      - ligne `7` : `if` :
        - ligne `8` : opération élémentaire : $O(1)$.
        - ligne `9` : opération élémentaire : $O(1)$.
        - **total :** $O(1)$.
    - **total :** $O(1) + T(p-1, q) + T(p, q-1)$.
- ligne `10` : opération élémentaire : $O(1)$.
- **total :**
  - cas d'arrêt $p = 0$ ou $q = 0$: $T(p, q) = O(1)$.
  - cas général $p > 0$ et $q > 0$: $T(p, q) = O(1) + T(p-1, q) + T(p, q-1)$.

- Utilisons le formulaire du cours pour résoudre la relation de récurrence :

  - Si on considère que p et q représentent des quantités symétriques, la complexité du cas général peut être simplifiée en $T(p,q) = O(1) + 2T(p-1,q)$
  - On se retrouve dans le cas suivant du formulaire du cours : $T(n) = cT(n-1) + \Theta(n^k)$ pour $c > 1$, avec ici $c = 2$ et $n = p + q$.
  - La complexité est donc en $O(c^n)$, c'est-à-dire ici $O(2^{p+q})$
- On peut aussi étudier ça de façon plus intuitive :
  - Le pire des cas est celui où les séquences n'ont aucun élément commun, car chaque appel de la fonction provoquera deux appels récursifs.
  - L'arbre d'appels alors obtenu est un arbre binaire dont la hauteur est $p + q$ (cf. exercice précédent).
  - Par conséquent, le nombre total d'appels (i.e. le nombre de nœuds dans l'arbre) est inférieur ou égal à 2^{p+q}.
  - Chaque appel correspond à un traitement en $O(1)$, donc la complexité temporelle du cas général est en $2^{p+q} \times O(1) = O(2^{p+q})$.
- On peut remarquer sur l'arbre d'appels que cette complexité très élevée est en partie due au fait qu'on effectue plusieurs fois les mêmes calculs. Par exemple, on traite deux fois la paire de séquences $(1,4)$ et (6). Il est possible que la complexité diminue si on modifie l'algorithme pour qu'il n'effectue plus ces calculs redondants : c'est le principe de la programmation dynamique.

4 Programmation dynamique

Exercice 5

La matrice L doit être initialisée dans le programme principal avec la valeur -1, ce qui va permettre de déterminer dans `plsc2` si une valeur $l_{i,j}$ a déjà été calculée ou pas.

```
int plsc2(int tab1[], int i, int tab2[], int j, int l[P+1][Q+1])
{  int resultat=0,resultat1,resultat2;

   if(l[i][j] == -1)
   {  if(i==0 && j==0)
         resultat = 0;
      else
      {  if(i>0)
            resultat1 = plsc2(tab1, i-1, tab2, j, l);
         else
            resultat1=0;
         if(j>0)
            resultat2 = plsc2(tab1, i, tab2, j-1, l);
         else
            resultat1 = 0;
         if(resultat1>resultat2)
            resultat = resultat1;
         else
            resultat = resultat2;
         //
         if(tab1[i-1] == tab2[j-1])
            resultat = 1 + plsc2(tab1, i-1, tab2, j-1, l);
      }
      l[i][j] = resultat;
   }
   else
      resultat = l[i][j];

   return resultat;
}
```

Exercice 6

- Dans l'approche par programmation dynamique, on utilise une matrice pour stocker les résultats intermédiaires, de manière à ne pas refaire un calcul qui a été déjà fait.
- Le nombre d'appels correspond donc à la taille de la matrice : $(p+1) \times (q+1)$.
- Comme la fonction effectue un traitement en temps constant, on a donc une complexité temporelle en $O(p \times q)$, soit une complexité asymptotique bien inférieure à celle de la version naïve de l'algorithme.
- **Remarque :** par contre, la complexité spatiale sera supérieure :
  - Dans la version naïve :
    - On utilise dans la fonction 3 variables entières et deux tableaux de tailles maximales respectives `P` et `Q`.
    - La hauteur de l'arbre étant $p+q$, on a une complexité en $O(3(p+q)+p+q) = O(p+q)$.
  - Dans la version améliorée, on a besoin d'une matrice $p \times q$, donc la complexité spatiale sera au moins de $O(pq)$.

5 Construction d'une PLSC

Exercice 7

Pour i et j tels que $l_{i,j} = k \neq 0$, soit $PLSC(X_i, Y_j) = (z_0, \dots, z_{k-1})$. On suppose (à démontrer par récurrence) la propriété suivante :

- Si $x_{i-1} = y_{i-1}$ alors $z_{k-1} = x_{i-1} = y_{i-1}$ et $(z_0, \dots, z_{k-2}) = PLSC(X_{i-1}, Y_{j-1})$.
- Si $x_{i-1} \neq y_{i-1}$ alors :
  - si $l_{i-1,j} > l_{i,j-1}$ alors $PLSC(X_i, Y_j) = PLSC(X_{i-1}, Y_j)$.
  - sinon $PLSC(X_i, Y_j) = PLSC(X_i, Y_{j-1})$.

Exercice 8

```
   void calcule_plsc(int l[P+1][Q+1],int tab1[P], int tab2[Q], int tab3[], int
taille3)
   {  int i=P,j=Q,k=taille3-1;

      while(i>0 && j>0)
      {  if(tab1[i-1] == tab2[j-1])
         {  tab3[k] = tab1[i-1];
            i--;
            j--;
            k--;
         }
         else
         {  if(l[i][j] == l[i-1][j])
               i--;
            else
               j--;
         }
      }
   }
```

Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

Correction TP 64 | arbres binaires

1 Arbres binaires

Exercice 1

```
arbre genere_arbre_complet(int h)
{  arbre resultat,g,d;
   int v;

   if(h==0)
      resultat = cree_arbre();
   else
   {  g = genere_arbre_complet(h-1);
      d = genere_arbre_complet(h-1);
      v = genere_entier(0,99);
      resultat = enracine(v, g, d);
   }
   return resultat;
}
```

Exercice 2

```
arbre genere_arbre_degenere(int h)
{  arbre resultat,g,d;
   int v;

   if(h==0)
      resultat = cree_arbre();
   else
   {  g = genere_arbre_degenere(h-1);
      d = cree_arbre();
      v = genere_entier(0,99);
      resultat = enracine(v, g, d);
   }
   return resultat;
}
```

Exercice 3

```
arbre genere_arbre_aleatoire(int n)
{  arbre resultat,g,d;
   int v,fils,ng=0,nd=0;

   if(n==0)
      resultat = cree_arbre();
   else
   {  fils = genere_entier(0,2);
      switch(fils)
      {  case 0:
            ng = n - 1;
            break;
         case 1:
            nd = n - 1;
            break;
         case 2:
            ng = (n-1) / 2;
            nd = (n-1) - ng;
            break;
```

```
        }
        g = genere_arbre_aleatoire(ng);
        d = genere_arbre_aleatoire(nd);
        v = genere_entier(0,99);
        resultat = enracine(v, g, d);
    }
    return resultat;
}
```

Exercice 4

```
int calcule_hauteur(arbre a)
{  int resultat = -1;
   int hg,hd;
   arbre fg,fd;

   if (!est_vide(a))
   {  fils_gauche(a,&fg);
      hg = calcule_hauteur(fg);
      fils_droit(a,&fd);
      hd = calcule_hauteur(fd);
      if(hg>hd)
         resultat = 1 + hg;
      else
         resultat = 1 + hd;
   }
   return resultat;
}
```

Exercice 5

```
int compte_noeuds(arbre a)
{  int resultat = 0;
   arbre fg,fd;

   if(!est_vide(a))
   {  // fils gauche
      fils_gauche(a,&fg);
      resultat = resultat + compte_noeuds(fg);
      // fils droit
      fils_droit(a,&fd);
      resultat = resultat + compte_noeuds(fd);
      // racine
      resultat++;
   }
   return resultat;
}
```

2 Arbres binaires de recherche

Exercice 6

```
void tableau_vers_arbre(int tab[N], arbre *a)
{  int i;
   for(i=0;i<N;i++)
      insere(a, tab[i]);
}
```

Exercice 7

```
void arbre_vers_tableau(arbre a, int tab[N], int *index)
{  int v;
   arbre g,d;

   if(!est_vide(a))
   {  // traitement du fils gauche
      fils_gauche(a,&g);
      arbre_vers_tableau(g,tab,index);
      // traitement de la racine
```

```
        racine(a,&v);
        tab[*index]=v;
        (*index)++;
        // traitement du fils droit
        fils_droit(a,&d);
        arbre_vers_tableau(d,tab,index);
    }
}
```

Exercice 8

```
void tri_arbre(int tab[N])
{  int index=0;
   arbre a = cree_arbre();

   tableau_vers_arbre(tab, &a);
   arbre_vers_tableau(a, tab, &index);
}
```