Structură și stil în PROGRAMARE

Bazil Pârv

Ultimul episod al serialului nostru conține detalii referitoare la modul în care ar trebui implementați diferiții algoritmi folosiți în rezolvarea unei anumite probleme. Vom exemplifica această modalitate pentru o problemă simplă și anume rezolvarea unei ecuații de gradul I.

În episodul anterior am prezentat implementarea mai multor algoritmi folosiți în programe. Nu am tratat elementele specifice problemei de rezolvat, și anume tipurile de date `TipDate` și `TipRezultate` și subprogramele `Antet`, `CitesteDate`, `CitesteDate2`, `Calculeaza`, `AfiseazaRezultate`.

Rezolvarea ecuației de gradul I

Enunțul acestei probleme este: *să se rezolve ecuația de gradul I: $a \cdot x + b = 0$.*

Specificarea sa este următoarea:

```
Specificarea EcGr1 este:     { varianta 0 }
  Date: a,b ∈ R
  Rezultate: s ∈ R astfel încât a * s + b = 0
SfSpecificare { EcGr1 }
```

Din păcate, specificarea de mai sus nu este completă, deoarece nu cuprinde toate situațiile posibile. Fiindcă nu am impus nici o condiție asupra coeficienților *a* și *b*, există două cazuri neacoperite. Discuția completă a ecuației cuprinde următoarele situații:

- $a \neq 0$: ecuația are soluția unică $s = -b / a$;
- $a = 0$ și $b \neq 0$: ecuația este incompatibilă (nu are nici o soluție);
- $a = 0$ și $b = 0$: ecuația este nedeterminată (are o infinitate de soluții).

Pentru a cuprinde toate situațiile de mai sus în specificarea problemei, introducem o variabilă de stare numită `cod`, cu următoarea semnificație:

- `cod = 0` pentru $a \neq 0$: ecuația are soluția unică $s = -b / a$;
- `cod = 1` pentru $a = 0$ și $b \neq 0$: ecuația este incompatibilă (nu are nici o soluție);
- `cod = 2` pentru $a = 0$ și $b = 0$: ecuația este nedeterminată (are o infinitate de soluții).

Cu aceste notații, specificarea de mai sus devine:

```
Specificarea EcGr1 este:     { varianta 1 }
  Date: a,b ∈ R
  Rezultate:
    cod = 0 și s ∈ R astfel încât a * s + b = 0
                                  dacă a ≠ 0 sau
    cod = 1 dacă a = 0 și b ≠ 0 sau
    cod = 2 dacă a = 0 și b = 0 sau
SfSpecificare { EcGr1 }
```

Din această specificare putem extrage componentele tipurilor de date `TipDate` și `TipRezultate` proiectate generic în cadrul episodului anterior:

```
type TipDate = record
       a,b:Real  { coeficientii ecuatiei de }
                 { gradul I: a * x + b = 0 }
     end; { TipDate }

     TipRezultate = record
       cod:Byte; { caracterizeaza solutia   }
         { cod = 0: solutie unica (a <> 0) }
         {                   memorata in s }
         { cod = 1: incompatibilitate      }
         {                 (a = 0, b <> 0) }
         { cod = 2: nedeterminare          }
         {                  (a = 0, b = 0) }
        s:Real { solutia ecuatiei -        }
               {    numai in cazul cod = 0 }
     end; { TipRezultate }
```

Implementări procedurale

Deoarece acum problema care trebuie rezolvată este specificată complet, putem să descriem și să implementăm

subalgoritmii Antet, CitesteDate, CitesteDate2, Calculeaza și AfiseazaRezultate. Prezentăm în continuare codurile sursă respective.

Implementarea subprogramului CitesteDate

```
procedure CitesteDate(var DI:TipDate);
{ citeste datele de intrare ale problemei
  parametru: DI - datele de intrare ale
                              problemei (OUT) }
begin
  Write('a=');
  Readln(DI.a);
  Write('b=');
  Readln(DI.b)
end; { CitesteDate }
```

Implementarea subprogramului CitesteDate2

```
procedure CitesteDate2(var DI:TipDate;
                       var SuntDate:Boolean);
{ citeste datele de intrare ale problemei
  (daca exista)
  parametri: DI - datele de intrare ale
                               problemei (OUT)
             SuntDate - specifica daca s-au
                citit datele de intrare (OUT)
             SuntDate = true - s-au citit
                           datele de intrare
             SuntDate = false - nu s-au citit
                         datele de intrare }
var c:Char;
begin
  repeat
    Write('Exista date de intrare (D/N): ');
    Readln(c)
  until c in ['D','d','N','n'];
  if c in ['D','d'] then
    begin
      SuntDate:=true;
      Write('a=');
      Readln(DI.a);
      Write('b=');
      Readln(DI.b)
    end
  else SuntDate:=false
end; { CitesteDate2 }
```

Implementarea subprogramului Antet

```
procedure Antet;
{ afiseaza antetul problemei de rezolvat }
begin
  Clrscr;
  Write('Programul calculeaza solutia ',
        'ecuatiei de gradul I a * x + b = 0')
end; { Antet }
```

Implementarea subprogramului Calculeaza

```
procedure Calculeaza(DI:TipDate;
                     var RC:TipRezultate);
{ calculeaza rezultatele RC pe baza datelor de
                                    intrare DI
  parametri: DI - datele de intrare ale
                                problemei (IN)
             RC - rezultatele calculate
                                     (OUT) }
begin
  if DI.a = 0 then
    if DI.b = 0 then
      RC.cod := 2
    else RC.cod := 1
  else
    begin
      RC.cod := 0;
      RC.s := -DI.b / DI.a
    end
end; { Calculeaza }
```

Implementarea subprogramului AfiseazaRezultate

```
procedure AfiseazaRezultate(DI:TipDate;
                            RC:TipRezultate);
{ afiseaza rezultatele si (eventual) datele de
                                    intrare
  parametri: DI - datele de intrare ale
                                problemei (IN)
             RC - rezultatele calculate (IN) }
begin
  Write('S-a rezolvat ecuatia: ');
  if DI.a <> 0 then
  begin
    Write(DI.a:10:2, '*X ');
    if DI.b > 0 then
      Write('+ ')
  end;
  Write(DI.b:10:2);
  Writeln(' = 0');
  case RC.cod of
    0: WriteLn('Ecuatia are solutia unica: ',
                                  RC.s:10:2);
    1: WriteLn('Ecuatia este incompatibila ',
                          '(a = 0, b <> 0)');
    2: WriteLn('Ecuatia este nedeterminata ',
                            '(a = 0, b = 0)')
  end
end; { AfiseazaRezultate }
```

Implementări modulare

Dacă studiem cele patru versiuni ale programelor principale, prezentate în cadrul episodului anterior, constatăm că declarațiile tipurilor de date TipDate și TipRezultate, precum și declarațiile subprogramelor Antet, CitesteDate, Calculeaza și AfiseazaRezultate se repetă în

fiecare dintre ele (cu excepția versiunii 2, în care CitesteDate este înlocuit cu CitesteDate2). Este natural, prin urmare, să proiectăm un modul care să conțină toate aceste declarații și apoi să-l folosim în toate versiunile programelor.

Un singur modul

O primă tentativă de modularizare (radicală) va include în modulul UEcGr1 toate declarațiile din programe, păstrând în acestea numai declarațiile de variabile globale și corpul programului principal. Interfața modulului UEcGr1 este:

```
Unit UEcGr1;
          { rezolvarea ecuatiei de gradul I }
interface
  type { declaratiile tipurilor de date     }
       {            TipDate si TipRezultate }
    TipDate = record
      a,b:Real  { coeficientii ecuatiei de }
                { gradul I: a * x + b = 0 }
    end; { TipDate }
    TipRezultate = record
      cod:Byte; { caracterizeaza solutia   }
         { cod = 0: solutie unica (a <> 0) }
         {                   memorata in s }
         { cod = 1: incompatibilitate      }
         {                 (a = 0, b <> 0) }
         { cod = 2: nedeterminare          }
         {                  (a = 0, b = 0) }
       s:Real { solutia ecuatiei -        }
              {    numai in cazul cod = 0 }
    end; { TipRezultate }
    { declaratia tipului de date TipOperatie }
    TipOperatie = (Terminare, Citire, Calcul,
                                      Afisare);

     { declaratiile subprogramelor publice }
  procedure Antet;
  procedure CitesteDate(var DI:TipDate);
  procedure CitesteDate2(var DI:TipDate;
                         var SuntDate:Boolean);
  procedure Calculeaza(DI:TipDate;
                       var RC:TipRezultate);
  procedure AfiseazaRezultate(DI:TipDate;
                              RC:TipRezultate);
  function Continuare:Boolean;
  procedure AfiseazaMeniu(var op:TipOperatie);
  procedure Mesaj(m:string);
```

Partea de declarații din programele prezentate în episodul anterior se va înlocui cu instrucțiunea:

```
uses UEcGr1;
```

Am obținut astfel programe generale, în care tot ce este specific problemei de rezolvat este "ascuns" într-un modul. Spre exemplu, varianta 3 are forma:

```
Program V3;
{ executie cu seturi repetate de date       }
{ continuarea sau terminarea executiei este }
{ separata de citirea datelor de intrare    }
uses UEcGr1;
          { rezolvarea ecuatiei de gradul I }
{ declaratia variabilelor globale }
var D:TipDate; { datele de intrare }
    R:TipRezultate; { rezultatele }
Begin
  repeat
    Antet; { subprogram de prezentare }
    CitesteDate(D);
          { subprogram care citeste datele }
    Calculeaza(D,R);
        { subprogram care face prelucrarile }
    AfiseazaRezultate(D,R)
     { subprogram care afiseaza rezultatele }
  until not Continuare
End. { V3 }
```

Înlocuind în fraza **uses** numele modulului UEcGr1 cu numele altui modul (în care este implementată rezolvarea altei probleme) programul va funcționa normal. Am obținut astfel patru variante generale de programe, în care am izolat particularitățile problemei de rezolvat într-un modul.

Două module

Dacă analizăm interfața modulului UEcGr1, constatăm că ea conține și declarații publice, care nu sunt specifice problemei de rezolvat: tipul de date TipOperatie (necesar pentru procedura AfiseazaMeniu) și subprogramele Continuare, AfiseazaMeniu și Mesaj.

Acestea sunt aceleași pentru orice problemă și, prin urmare, este natural să le punem în alt modul, pe care îl vom numi UService.

Prin urmare, am "spart" modulul UEcGr1 în două module mai mici: UService (care este general) și UEcGr1V2 (care va conține numai elementele specifice rezolvării problemei, adică tipurile de date TipDate și TipRezultate și subprogramele Antet, CitesteDate, Calculeaza și AfiseazaRezultate.

Interfața modulului UService este:

```
Unit UService;
interface
   { declaratia tipului de date TipOperatie }
  type TipOperatie = (Terminare, Citire,
                              Calcul, Afisare);

     { declaratiile subprogramelor publice }
  function Continuare:Boolean;
  procedure AfiseazaMeniu(var op:
                                  TipOperatie);
  procedure Mesaj(m:string);
```

În interfața modulului UEcGr1V2 vor rămâne numai declarațiile.

```
Unit UEcGr1;
          { rezolvarea ecuatiei de gradul I }
interface
  type { declaratiile tipurilor de date     }
       {            TipDate si TipRezultate }
    TipDate = record
      a,b:Real  { coeficientii ecuatiei de }
                { gradul I: a * x + b = 0 }
    end; { TipDate }
    TipRezultate = record
      cod:Byte; { caracterizeaza solutia  }
         { cod = 0: solutie unica (a <> 0) }
         {                   memorata in s }
         { cod = 1: incompatibilitate      }
         {                 (a = 0, b <> 0) }
         { cod = 2: nedeterminare          }
         {                  (a = 0, b = 0) }
       s:Real { solutia ecuatiei -         }
              {    numai in cazul cod = 0 }
    end; { TipRezultate }

     { declaratiile subprogramelor publice }
  procedure Antet;
  procedure CitesteDate(var DI:TipDate);
  procedure CitesteDate2(var DI:TipDate;
                         var SuntDate:Boolean);
  procedure Calculeaza(DI:TipDate;
                         var RC: TipRezultate);
  procedure AfiseazaRezultate(DI:TipDate;
                              RC:TipRezultate);
```

Părțile de declarații din programe se vor înlocui cu instrucțiunea **uses** UEcGr1V2; (pentru primele două versiuni), respectiv **uses** UEcGr1V2,UService; (pentru celelalte două versiuni).

Noua arhitectură obținută izolează și mai bine specificul problemei de rezolvat de programele generale obținute. Pe lângă scheletul acestora (în care singurul parametru este numele modulului care conține rezolvarea problemei), am obținut și un modul general UService, care conține proceduri de serviciu, necesare pentru ultimele două versiuni ale programelor.

Concluzii

Oare de câte ori au rostit elevi și studenți: "*De ce nu e bine? Doar funcționează!*"?

Da, stimați cititori, este nevoie de încă ceva: programele nu se scriu ***oricum*** - se proiectează astfel încât oricând ar dori autorul programului respectiv, sau un alt programator să-l "priceapă" repede, să poată refolosi părți din el, să-l poată folosi în condiții diferite, operând un număr cât mai mic de modificări.

Dacă un program nu este scris respectând paradigma programării modulare, nu există "părți", dacă nu se ține cont de necesitatea scrierii unor programe flexibile și parametrizate, reutilizarea poate să devină imposibilă iar orice modificare va însemna, de fapt, scrierea unui program complet nou.

În aceste condiții etapele care trebuie respectate în realizarea unui program în condiții de dezvoltare incrementală vor fi:

1. Scrierea unor programe ținând cont de cerința de funcționalitate. Mai întâi se vor proiecta algoritmi (subalgoritmi) cât mai simpli; se va pune accent pe structura programului. Funcționalitatea nu poate fi realizată decât dacă se respectă cerințele impuse de problema de rezolvat.
2. Dacă prima versiune a algoritmilor proiectați nu se dovedesc a fi suficient de performanți, respectiv, permit optimizări, acestea se vor căuta și se vor implementa.
3. Nu pot fi ignorate acele modificări necesare de "înfrumusețare" care asigură o utilizare ușoară chiar și pentru un "beneficiar neinstruit" printr-o interfață prietenoasă, realizată și aceasta cât se poate de simplu.

Dl. prof. univ. Bazil Pârv este cadru didactic la Universitatea "Babeș-Bolyai" din Cluj. Poate fi contactat prin e-mail la **bparv@cs.ubbcluj.ro**.

Compania Microsoft a combinat puterea a trei dintre sistemele de operare proprii pentru a crea următoarea generație de sistemele de operare Windows.

Denumirea noului produs provine de la denumirile celor trei produse ale căror caracteristici au fost folosite: Windows CE, Windows Me și Windows NT. Astfel, numele noului sistem de operare este Windows CEMeNT.

Numele descrie o parte dintre caracteristicile produsului: *robustețea unei stânci și "inteligența" unei cărămizi*.