Structură și stil în PROGRAMARE

Bazil Pârv

Limbajul Pascal a fost creat de Niklaus Wirth în scopuri didactice pentru a oferi celor care învață programare un limbaj care să faciliteze respectarea principiilor programării structurate. Acest limbaj, la ora actuală, este cel mai răspândit limbaj de programare care se regăsește în programele școlare. Cu toate acestea există o serie de cunoștințe fundamentale cărora nu li se acordă suficientă atenție pe parcursul studiului în școli. Mulți elevi, dar și dascălii lor se mulțumesc nu odată, dacă programul "merge" și nu țin cont de eficiență sau de stil. Vom încerca, într-o serie de articole să reliefăm câteva din aceste aspecte.

Structura unui program Pascal

Structura unui program *Pascal* este un lucru binecunoscut, așadar o reamintim doar:

```
Program nume_program; { antetul programului }
  declarații
Begin       { instructiunea compusa (corpul) }
                  { programului nume_program }
    instr_1;
    ...
    instr_n
End.    { sfarsitul programului nume_program }
```

Partea de *declarații* poate să fie vidă sau poate să conțină declarații, respectiv definiții de:

- constante (**const**)
- tipuri de date (**type**)
- etichete (**label**)
- subprograme (**procedure** sau **function**)
- variabile globale (**var**)

În condiții *normale* un program este soluția unei probleme. În cele ce urmează vom prezenta câteva tipare de structurare a programelor *Pascal* care, în concepția noastră, ar trebui să respecte următoarele cerințe:

- structura programului principal va fi independentă de problema de rezolvat;
- elementele specifice ale problemei de rezolvat vor fi "ascunse" în subprograme;
- corpul programului principal va conține numai apeluri de subprograme.

O analiză independentă de problema care se rezolvă

Pentru orice problemă *P* pe care dorim s-o rezolvăm, notăm cu *D* mulțimea datelor de intrare și cu *R* mulțimea rezultatelor. Cu aceste notații, specificarea problemei se face astfel:

```
Specificarea P este:
  Date: D
  Rezultate: R
SfSpecificare { P }
```

Pe baza acestei specificări, vom discuta trei modele de structură a algoritmului de rezolvare a problemei, care corespund situațiilor:

- programul prevede execuție pentru un singur set de date;
- programul se va executa pentru mai multe seturi de date;
- execuția poate fi "controlată" cu ajutorul unui meniu.

Execuție cu un singur set de date

Într-o exprimare naturală, rezolvarea problemei *P* se poate descrie algoritmic în felul următor:

```
Algoritmul P este:          { versiunea 0 }
  @Citește datele de intrare D
  @Calculează rezultatele R pe baza datelor de intrare D
```

```
   @Afișează rezultatele R (și eventual datele de intrare D)
SfAlgoritm { P }
```

Descrierea de mai sus nu este completă, deoarece conține numai propoziții nestandard (prefixate de caracterul '@'). Ea exprimă modalitatea de rezolvare a problemei *P* prin descompunerea acesteia în trei subprobleme, fiecare mai simplă decât *P*. Soluția problemei *P* se obține prin combinarea soluțiilor subproblemelor (în acest caz este vorba de compunere secvențială).

Fiecare subproblemă rezultată din descompunere poate fi considerată subalgoritm. Folosim următoarele notații:

- `CitesteDate(DI)` - subalgoritmul corespunzător subproblemei *@Citește datele de intrare* `DI`
- `Calculeaza(DI,RC)` - subalgoritmul corespunzător subproblemei *@Calculează rezultatele* `RC` *pe baza datelor de intrare* `DI`
- `AfiseazaRezultate(DI,RC)` - subalgoritmul corespunzător subproblemei *@Afișează rezultatele* `RC` *(și eventual datele de intrare* `DI`*)*

Cu aceste notații, versiunea `0` a algoritmului *P* devine:

```
Algoritmul P este:                { versiunea 1 }
  Apelează CitesteDate(D)
  Apelează Calculeaza(D, R)
  Apelează AfiseazaRezultate(D, R)
SfAlgoritm { P }
```

Subalgoritmii de mai sus sunt specificați în cele ce urmează. Singurele elemente care se precizează sunt *precondițiile*, *postcondițiile*, *semnificațiile* și *tipurile parametrilor* (**IN** - de intrare, **OUT** - de ieșire) și scopul subalgoritmului.

```
SubAlgoritmul CitesteDate(DI) este:
  Pre: true
  Post: DI au fost citite
  Parametri:
    DI - datele de intrare ale problemei P (OUT)
  Scop:
    Citește DI de la intrarea standard
SfSubAlgoritm { CitesteDate }

SubAlgoritmul Calculeaza(DI,RC) este:
  Pre: DI au fost citite
  Post: RC au fost calculate
  Parametri:
    DI - datele de intrare ale problemei P (IN)
    RC - rezultatele problemei P (OUT)
  Scop:
    Calculează RC pe baza DI
SfSubAlgoritm { Calculeaza }

SubAlgoritmul AfiseazaRezultate(DI,RC) este:
  Pre: DI au fost citite
       RC au fost calculate
  Post: true
  Parametri:
    DI - datele de intrare ale problemei P (IN)
    RC - rezultatele problemei P (IN)
  Scop:
    Afișează RC (și eventual DI) la ieșirea standard
SfSubAlgoritm { AfiseazaRezultate }
```

Specificarea acestor subalgoritmi conține, pe lângă elementele uzuale, precondițiile și postcondițiile acestora. ***Precondiția*** este un predicat care specifică condițiile în care algoritmul se poate executa (el exprimă condițiile impuse asupra parametrilor de intrare), iar ***postcondiția*** este un predicat care este adevărat la terminarea execuției (el exprimă legătura dintre parametrii de intrare și de ieșire). Să remarcăm că ordinea de apelare a subalgoritmilor din versiunea `1` a algoritmului *P* asigură automat verificarea precondițiilor subalgoritmilor respectivi.

Am ajuns astfel la o primă versiune finală a algoritmului *P*, care conține numai apeluri de subalgoritmi. Această versiune este generală, deoarece caracteristicile problemei *P* sunt "ascunse" în:

- datele de intrare `D` și rezultatele `R`, care nu au fost explicitate;
- subalgoritmii `CitesteDate`, `Calculeaza` și `AfiseazaRezultate`, care au fost doar specificați, nu și descriși.

Atât structurile de date (pentru datele de intrare `D` și rezultatele `R`), cât și subalgoritmii enumerați depind de problema *P* care trebuie rezolvată. Analiza lor se amână până în momentul când vom considera un exemplu.

În cele ce urmează vom discuta alte moduri de combinare a soluțiilor subproblemelor pentru a obține versiuni structural diferite ale algoritmului *P*.

Execuție cu mai multe seturi de date

Versiunea `1` a algoritmului *P* corespunde unei execuții cu un singur set de date. Dacă dorim să repetăm execuția cu un alt set de date, programul trebuie lansat din nou în execuție.

Pentru a permite execuția programului cu mai multe seturi de date, următoarele două versiuni ale algoritmului *P* includ cele trei subprobleme (în aceeași ordine) într-o structură repetitivă. Descrierea inițială a algoritmului în acest caz este:

```
Algoritmul P este:                { versiunea 0R }
  Repetă
    @Citește datele de intrare D
    @Calculează rezultatele R pe baza datelor de intrare D
    @Afișează rezultatele R (și eventual datele de intrare D)
  PânăCând @Nu mai sunt date de intrare
SfAlgoritm { P }
```

Constatăm aici un alt mod de compunere a soluțiilor subproblemelor pentru a rezolva problema inițială: folosirea structurii repetitive **Repetă**...**PânăCând**. De asemenea, să remarcăm că a apărut o subproblemă nouă: *@Nu*

mai sunt date de intrare, care reprezintă criteriul de terminare a execuției programului.

Pentru această subproblemă avem cel puțin două variante de discuție:

- faptul că nu mai sunt date de intrare se stabilește în subproblema *@Citește datele de intrare* `D`;
- continuarea execuției cu un alt set de date de intrare este separată de citirea datelor de intrare.

Continuarea execuției pe baza datelor de intrare

Versiunea `0R` a descrierii algoritmului devine în această situație:

```
Algoritmul P este:                    { versiunea 0R1 }
  Repetă
    @Citește datele de intrare D (dacă mai sunt date de
      intrare)
    dacă @Mai sunt date de intrare
      atunci
        @Calculează rezultatele R pe baza datelor de in-
          trare D
        @Afișează rezultatele R (și eventual datele de
          intrare D)
    SfDacă
  PânăCând @Nu mai sunt date de intrare
SfAlgoritm { P }
```

În această versiune constatăm următoarele:

- s-a modificat subproblema aferentă citirii datelor de intrare (ea trebuie să decidă dacă mai sunt date de citit și să le citească numai în caz afirmativ);
- execuția ultimelor două subprobleme inițiale este condiționată de existența datelor de intrare (specificată prin noua subproblemă *@Mai sunt date de intrare*);
- subproblemele *@Mai sunt date de intrare* și *@Nu mai sunt date de intrare* exprimă situații opuse una celeilalte (când prima este adevărată, a doua este falsă și invers).

Să remarcăm că acum soluția problemei *P* care se obține prin compunerea soluțiilor subproblemelor inițiale, folosește toate structurile de calcul proprii programării structurate: *secvența*, *repetiția* și *decizia*. Operația de compunere a soluțiilor subproblemelor pentru a obține soluția problemei inițiale are acest înțeles: ***secvență*** (soluția unei subprobleme este folosită ca dată de intrare în următoarea problemă ce apare în secvență), ***repetiție*** (grupurile de soluții parțiale se pot repeta), ***decizie*** (o anumită subproblemă se rezolvă numai dacă este îndeplinită o anumită condiție) și ***apel recursiv*** (aceeași subproblemă de rezolvat, cu date de complexitate sau valoare mai mică).

Ținând cont de observațiile de mai sus, descrierea finală a algoritmului *P* este:

```
Algoritmul P este:                    { versiunea 1R1 }
  Repetă
    Apelează CitesteDate2(D,SuntDate)
    Dacă SuntDate
      atunci
        Apelează Calculeaza(D,R)
        Apelează AfiseazaRezultate(D,R)
    SfDacă
  PânăCând not SuntDate
SfAlgoritm { P }
```

Pentru acest algoritm trebuie precizate următoarele:

- subalgoritmii `Calculeaza` și `AfiseazaRezultate` au specificarea prezentată anterior în cadrul acestui articol;
- am proiectat un subalgoritm nou: `CitesteDate2`, corespunzător noii subprobleme de citire a datelor de intrare;
- subproblemele *@Mai sunt date de intrare* și *@Nu mai sunt date de intrare* sunt reprezentate printr-o variabilă locală `SuntDate`, obținută ca parametru de ieșire din `CitesteDate2`.

Remarcăm că această variantă de structură a algoritmului folosește numai parțial subalgoritmii prezentați în cazul execuției cu un singur set de date.

Ca observație generală, este bine să respectăm principiul: *orice algoritm rezolvă o subproblemă bine definită*. În cazul nostru, `CitesteDate2` are două sarcini: atât decizia privitoare la citirea următorului set de date, cât și citirea propriu-zisă a acestuia. Este de preferat să separăm citirea datelor de decizia de continuare. Acest articol va conține și prezentarea unei variante care realizează acest lucru. Specificarea noului subalgoritm `CitesteDate2` este:

```
SubAlgoritmul CitesteDate2(DI,ExistaDate) este:
  Pre: true
  Post: DI au fost citite iar ExistaDate a fost setat
  Parametri:
    DI - datele de intrare ale problemei P (OUT)
    ExistaDate - indicator al existenței datelor citite
                 (OUT)
      ExistaDate = true dacă s-au citit datele de
                        intrare
      ExistaDate = false în caz contrar
  Scop:
    Citește DI de la intrarea standard (dacă există) și
    setează ExistaDate
SfSubAlgoritm { CitesteDate2 }
```

Versiunea `1R1` este o altă versiune finală a algoritmului *P*. Toate observațiile făcute la sfârșitul secțiunii referitoare la algoritmii care se execută pentru un singur set de date sunt valabile și aici: precondițiile subalgoritmilor sunt verificate prin ordinea de apelare a acestora, algoritmul este general, iar toate detaliile referitoare la problema *P* urmează a fi discutate la proiectarea subalgoritmilor.

Continuarea execuției separată de intrare

Această variantă de execuție cu mai multe seturi de date are ca punct de plecare descrierea incompletă `0R2`, care este aproape identică cu `0R`:

```
Algoritmul P este:                    { versiunea 0R2 }
  Repetă
     @Citește datele de intrare D
     @Calculează rezultatele R pe baza datelor de intrare D
     @Afișează rezultatele R (și eventual datele de intrare D)
  PânăCând not @Se continuă cu un alt set de date
SfAlgoritm { P }
```

În descrierea de mai sus, celor trei subprobleme inițiale li se adaugă o a patra, *@Se continuă cu alt set de date*, căreia îi va corespunde un subalgoritm numit `Continuare`. Varianta finală a descrierii algoritmului *P* este:

```
Algoritmul P este:                    { versiunea 1R2 }
  Repetă
    Apelează CitesteDate2(D,SuntDate)
    Apelează Calculeaza(D,R)
    Apelează AfiseazaRezultate(D,R)
  PânăCând not Continuare
SfAlgoritm { P }
```

Se observă că
- subalgoritmii `CitesteDate`, `Calculeaza` și `AfiseazaRezultate` au specificarea prezentată anterior;
- am proiectat un subalgoritm nou: `Continuare`, corespunzător noii subprobleme de continuare a citirii datelor de intrare.

Subalgoritmul `Continuare` este independent de problema *P*. Deoarece este suficient de simplu, vom preciza direct descrierea sa finală:

```
SubAlgoritmul Continuare este:
  Afișeaza "Continuati (D/N)? ";
  Citește raspuns;
  Dacă raspuns = "D"
    atunci Returneaza true
    altfel Returneaza false
  SfDacă
SfSubAlgoritm { Continuare }
```

Remarcăm că această variantă de rezolvare folosește toți subalgoritmii identificați în secțiunea referitoare la algoritmii executați pentru un singur set de date de intrare, adăugând unul nou, `Continuare`, necesar execuției repetate.

Versiunea `1R2` este a treia versiune finală a algoritmului *P*. Toate observațiile anterioare sunt valabile și aici: precondițiile subalgoritmilor sunt verificate automat prin ordinea în care apar apelurile acestora, algoritmul este general, toate detaliile referitoare la problema *P* urmează a fi discutate la proiectarea subalgoritmilor. În plus, se apelează subalgoritmul `Continuare` care este independent de *P*.

Execuție dirijată de meniu

Variantele de algoritmi prezentate până acum presupun o rezolvare liniară a problemei, în sensul că după ce s-au introdus datele de intrare restul procesului este realizat în secvență, fără intervenția utilizatorului. Mai exact, subalgoritmii `Calculeaza` și `AfiseazaRezultate` se apelează (în această ordine) numai dacă s-au citit datele de intrare, asigurându-ne că precondițiile lor sunt verificate automat.

Ultima variantă a algoritmului, pe care o discutăm aici, este structural diferită de celelalte și dă posibilitatea utilizatorului să comande fiecare pas din execuția programului după dorință, inclusiv terminarea acestuia:

```
Algoritmul P este:                    { versiunea 0M }
  Repetă
    @Afișează meniul și citește operația selectată de către
      utilizator din meniu
    Dacă operație = Citire
      atunci @Citește datele de intrare D
      altfel
        Dacă operație = Calcul
          atunci @Calculează rezultatele R pe baza
                   datelor de intrare D
          altfel
            Dacă operație = Afișare
              atunci @Afișează rezultatele R (și e-
                       ventual datele de intrare D)
            SfDacă
        SfDacă
    SfDacă
  PânăCând operație = Terminare
SfAlgoritm { P }
```

În descrierea de mai sus, celor trei subprobleme inițiale li se adaugă o a patra, *@Afișează meniul și citește operația selectată de către utilizator din meniu*, căreia îi va corespunde un subalgoritm numit `AfiseazaMeniu`. Înlocuind propozițiile nestandard cu apeluri de subalgoritmi obținem varianta următoare a descrierii algoritmului *P*:

```
Algoritmul P este:                    { versiunea 1M }
  Repetă
    Apelează AfiseazaMeniu(operatie)
    Dacă operație = Citire
      atunci
        Apelează CitesteDate(D)
      altfel
        Dacă operație = Calcul
          atunci
            Apelează Calculeaza(D,R)
          altfel
            Dacă operație = Afișare
              atunci
                Apelează AfiseazaRezultate
                                        (D,R)
            SfDacă
        SfDacă
    SfDacă
  PânăCând operație = Terminare
SfAlgoritm { P }
```

Remarcăm că
- subalgoritmii `CitesteDate`, `Calculeaza` și `AfiseazaRezultate` au specificarea prezentată anterior;
- am proiectat un subalgoritm nou: `AfiseazaMeniu`, corespunzător noii subprobleme de afișare a meniului și de citire a opțiunii utilizatorului.

Subalgoritmul `AfiseazaMeniu` este independent de problema *P*. El are un parametru de ieșire ce reprezintă codul operației cerute de utilizator. Deoarece este suficient de simplu, vom preciza direct descrierea sa finală:

```
SubAlgoritmul AfiseazaMeniu(operatie) este:
  Repetă
    Afișează "Operatiile posibile sunt:"
    Afișează "1.Citirea datelor de intrare"
    Afișează "2.Calcularea rezultatelor"
    Afișează "3.Afisarea rezultatelor"
    Afișează "0.Terminarea executiei
              programului"
    Afișează "Introduceti optiunea (0-3): "
    Citește cod
  PânăCând cod în {0, 1, 2, 3}
  Selectează cod dintre
    1:operație:=Citire
    2:operație:=Calcul
    3:operație:=Afisare
    0:operație:=Terminare
  SfSelectează
SfSubAlgoritm { AfiseazaMeniu }
```

Remarcăm folosirea structurii de selectare **Selectează...SfSelectează** în locul unor structuri alternative în cascadă.

Examinând atent versiunea `1M` a algoritmului problemei *P* constatăm că ea este incorectă, în sensul că nu se mai păstrează ordinea firească de apelare a subalgoritmilor: `Calculeaza` după `CitireDate`, respectiv `AfisareRezultate` după `Calculează`. Cu alte cuvinte, nu sunt respectate precondițiile algoritmilor respectivi, ei putând fi apelați în orice ordine. Prin urmare, înainte de apelarea lor în algoritmul *P*, trebuie să le verificăm precondițiile. Noua versiune a algoritmului - de data aceasta finală - arată astfel:

```
Algoritmul P este:                { versiunea 2M }
  ExistaDate:=false
  ExistaRezultate:=false
  Repetă
    Apelează AfiseazaMeniu(operatie)
    Selectează operatie dintre
      Citire: Apelează CitesteDate(D)
              ExistaDate := true
              ExistaRezultate := false
      Calcul: Dacă ExistaDate
                atunci
                  Apelează Calculeaza(D,R)
                  ExistaRezultate:=true
                altfel
                  Afișează "Nu s-au citit da-
                            tele de intrare"
              SfDacă
      Afisare: Dacă ExistaRezultate
                atunci
                  Apelează
                    AfiseazaRezultate(D, R)
                altfel
                  Afișează "Nu s-au calcu-
                            lat rezultatele"
              SfDacă
    SfSelectează
  PânăCând operatie = Terminare
SfAlgoritm { P }
```

Am prevăzut doi indicatori de stare pentru verificarea precondițiilor, `ExistaDate` (pentru `Calculeaza`) și `ExistaRezultate` (pentru `AfiseazaRezultate`). Inițial, aceștia sunt setați pe `false` și
- la operația `Citire` (care nu este condiționată) `ExistaDate` se setează pe `true`, iar `ExistaRezultate` pe `false`;
- la operația `Calcul` se apelează `Calculeaza` numai dacă `ExistaDate`, caz în care se setează `ExistaRezultate` pe `true`;
- la operația `Afisare` se apelează `AfisareRezultate` numai dacă `ExistaRezultate`;
- în toate cazurile când precondițiile nu sunt verificate se afișează mesaje de eroare.

De asemenea, remarcăm înlocuirea structurilor **Dacă...SfDacă** în cascadă din versiunea `1M` cu structura de selecție **Selectează...SfSelectează**, fapt care îmbunătățește exprimarea algoritmului și ușurează înțelegerea sa prin reducerea nivelurilor de indentare.

Versiunea `2M` este ultima versiune finală a algoritmului *P*. Aproape toate observațiile făcute la sfârșitul secțiunii referitoare la execuția pentru un singur set de date sunt valabile și aici: algoritmul este general, iar toate detaliile referitoare la problema *P* urmează a fi discutate la proiectarea subalgoritmilor. De data aceasta, precondițiile subalgoritmilor sunt verificate în corpul algoritmului folosind indicatori de stare. În plus, se apelează subalgoritmul `AfiseazaMeniu` care este independent de *P*.

În numerele următoare

Serialul nostru va continua cu prezentarea modului de proiectare și implementare a algoritmilor generali. Vom prezenta proiectarea tipurilor de date, precum și variante ale algoritmului general de rezolvare a unei probleme. De asemenea vor fi descrise și modurile de implementare a elementelor specifice unei probleme.

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