TUDOR SORIN

Manual de INFORMATICĂ

pentru clasa a X-a

Pascal şi C++

TEHNICI DE PROGRAMARE

PROFILUL REAL - INTENSIV

Respectă programa şcolară actuală!

Conţinutul este aprobat MEN prin ordinul nr. 4606/83 din 18.07.2005
şi MEdC prin nr. 4446 din 19.06.2006

L&S Soft

Cuprins

Capitolul 1. Subprograme 7

1.1. Noţiunea de subprogram 7
1.2. Subprograme în Pascal 9
1.2.1. Un exemplu de utilizare a funcţiilor 9
1.2.2. Un exemplu de utilizare a procedurilor 11
1.2.3. Structura unui subprogram 12
1.2.3.1. Structura subprogramelor de tip funcţie 12
1.2.3.2. Structura subprogramelor de tip procedură 13
1.2.4. Definirea şi declararea unui subprogram 14
1.2.5. Apelul subprogramelor 17
1.2.5.1. Apelul funcţiilor 17
1.2.5.2. Apelul procedurilor 18
1.2.5.3. Transmiterea parametrilor la apel 18
1.2.5.4. Cum memorează subprogramele parametrii trimişi? 20
1.2.5.5. Transmiterea parametrilor prin valoare 21
1.2.5.6. Transmiterea parametrilor prin referinţă 22
1.2.6. Variabile locale şi globale 23
1.2.7. Greşeli frecvente 25
1.2.8. Unităţi de program 26
1.3. Subprograme în C++ 29
1.3.1. Exemple de utilizare a funcţiilor 29
1.3.2. Structura unei funcţii 31
1.3.3. Declararea variabilelor 33
1.3.4. Transmiterea parametrilor 36
1.3.5. Definirea şi declararea unui subprogram 40
1.4. Aplicaţii care folosesc subprograme 42
Probleme propuse 49
Răspunsuri 59

Capitolul 2. Şiruri de caractere 60

2.1. Generalităţi 60
2.2. Şiruri de caractere în Pascal 61
2.2.1. Noţiuni introductive 61
2.2.2. Concatenarea şirurilor 64
2.2.3. Compararea şirurilor 65
2.2.4. Lungimea şirurilor de caractere 66

2.2.5. Subşiruri ... 67
2.2.6. Conversii de la şiruri la valori numerice şi invers ... 71
2.2.7. Citirea şi scrierea datelor de tip *String* din şi în fişiere text ... 75

2.3. Şiruri de caractere în C++ ... 76
2.3.1. Generalităţi ... 76
2.3.2. Citirea şi scrierea şirurilor de caractere ... 76
2.3.3. Tipul char* ... 79
2.3.4. Lungimea unui şir de caractere ... 80
2.3.5. Copierea şi concatenarea şirurilor de caractere ... 81
2.3.6. Căutarea unui caracter într-un şir ... 82
2.3.7. Compararea şirurilor ... 84
2.3.8. Subşiruri ... 86
2.3.9. Alte funcţii utile în prelucrarea şirurilor ... 88
2.3.10. Conversia şirurilor în valori numerice şi invers ... 91
2.3.11. Citirea şi scrierea şirurilor de caractere din şi în fişiere text ... 95
2.3.11.1. Operaţia de citire ... 95
2.3.11.2. Operaţia de scriere ... 96
2.3.12. O modalitate de conversie de la şir la alt tip ... 96
Probleme propuse ... 97

Capitolul 3. Structuri de date neomogene ... 99

3.1. Noţiuni introductive ... 99
3.2. Structuri neomogene în Pascal ... 99
3.2.1. Tipul *Record* ... 99
3.2.2. Accesul simplificat la câmpuri ... 101
3.2.3. Înregistrări imbricate ... 102
3.2.4. Vectori de înregistrări ... 102
3.2.5. Înregistrare cu variante ... 103
3.3. Structuri neomogene în C++ ... 105
3.3.1. Tipul *struct* ... 105
3.3.2. Înregistrări imbricate ... 107
3.3.3. Înregistrări cu structură variabilă ... 108
Probleme propuse ... 110

Capitolul 4. Structuri de date ... 111

4.1. Conceptul de structură de date ... 111
4.2. Structura de tip listă liniară ... 113
4.2.1. Prezentarea structurii ... 113
4.2.2. Liste alocate secvenţial ... 114
4.2.3. Liste alocate înlănţuit ... 115
4.2.4. Implementarea alocării înlănţuite prin utilizarea vectorilor ... 116
4.3. Structura de tip stivă ... 120
4.4. Structura de tip coadă ... 125
Probleme propuse ... 125
Răspunsuri ... 127

Capitolul 5. Alocarea dinamică a memoriei 128

5.1. Generalităţi 128
5.2. Variabile de tip pointer 129
5.2.1. Variabile de tip pointer în Pascal 129
5.2.2. Variabile de tip pointer în C++ 132
5.3. Alocarea dinamică a memoriei 135
5.3.1. Alocarea dinamică în Pascal 135
5.3.2. Alocarea dinamică în C++ 138
Probleme propuse 142
Răspunsuri 143

Capitolul 6. Structuri de date alocate dinamic 144

6.1. Liste liniare 144
6.2. Liste liniare alocate simplu înlănţuit 145
6.2.1. Prezentare generală 145
6.2.2. Crearea şi afişarea listelor 145
6.2.3. Operaţii asupra unei liste liniare 149
6.2.4. Aplicaţii ale listelor liniare 155
6.2.4.1. Sortarea prin inserţie 155
6.2.4.2. Sortarea topologică 157
6.2.4.3. Operaţii cu polinoame 162
6.3. Liste liniare alocate dublu înlănţuit 171
6.3.1. Crearea unei liste liniare alocată dublu înlănţuit 171
6.3.2. Adăugarea unei înregistrări la dreapta 172
6.3.3. Adăugarea unei înregistrări la stânga 172
6.3.4. Adăugarea unei înregistrări în interiorul listei 172
6.3.5. Ştergerea unei înregistrări din interiorul listei 173
6.3.6. Ştergerea unei înregistrări la stânga/dreapta listei 174
6.3.7. Listarea de la stânga la dreapta listei 174
6.3.8. Listarea de la dreapta la stânga listei 174
6.4. Liste circulare 177
6.5. Stiva implementată ca listă liniară simplu înlănţuită 177
6.6. Coada implementată ca listă liniară simplu înlănţuită 178
Probleme propuse 180
Răspunsuri la testele grilă 185

Capitolul 7. Introducere în recursivitate 186

7.1. Prezentare generală 186
7.2. Modul în care se realizează autoapelul 186
7.2.1. Realizarea autoapelului în Pascal 186
7.2.2. Realizarea autoapelului în C++ 187
7.3. Mecanismul recursivităţii 188
7.4. Cum gândim un algoritm recursiv? 192
7.5. Aplicaţii recursive 193
7.5.1. Aplicaţii la care se transcrie o formulă recursivă 193
7.5.2. Aplicaţii la care nu dispunem de o formulă de recurenţă 198
Probleme propuse 204
Indicaţii / Rezolvări 211

Capitolul 8. Metoda Divide et Impera 217

8.1. Prezentare generală 217
8.2. Aplicaţii 217
8.2.1. Valoarea maximă dintr-un vector 217
8.2.2. Sortarea prin interclasare 219
8.2.3. Sortarea rapidă 221
8.2.4. Turnurile din Hanoi 224
8.2.5. Problema tăieturilor 225
8.3. Fractali 228
8.3.1. Elemente de grafică 228
8.3.1.1. Generalităţi (varianta Pascal) 228
8.3.1.2. Generalităţi (varianta C++) 230
8.3.1.3. Setarea culorilor şi procesul de desenare 231
8.3.2. Curba lui Koch pentru un triunghi echilateral 233
8.3.3. Curba lui Koch pentru un pătrat 236
8.3.4. Arborele 238
Probleme propuse 240
Răspunsuri 241

Capitolul 9. Complexitatea algoritmilor 243

9.1. Exprimarea complexităţii 243
9.2. Ce trebuie să mai ştim ? 248
Probleme propuse 248
Răspunsuri 250

Anexa 1. Memento 251

A.1. Limbajul Pascal 251
A.1.1. Tipuri standard 251
A.1.2. Constante 252
A.1.3. Operatori 252
A.1.3.1. Prioritatea operatorilor 252
A.1.3.2. Operatori aritmetici 253
A.1.3.3. Operatori relaţionali 254
A.1.3.4. Operatori logici 254
A.1.4. Instrucţiuni 255
A.1.5. Câteva funcţii utile 258
A.2. Limbajul C++ 260
A.2.1. Tipuri standard 260
A.2.2. Constante 260
A.2.3. Operatori 262
A.2.3.1. Prioritatea operatorilor 262
A.2.3.2. Operatori aritmetici 262
A.2.3.3. Operatori relaţionali 263
A.2.3.4. Operatori de egalitate 263
A.2.3.5. Operatori de incrementare şi decrementare 264
A.2.3.6. Operatori logici 264
A.2.3.7. Operatori logici pe biţi 264
A.2.3.8. Operatori de atribuire 265
A.2.3.9. Operatorul "," 266
A.2.3.10. Operatorul condiţional 266
A.2.3.11. Operatorul "sizeof" 267
A.2.3.12. Operatorul de conversie explicită 267
A.2.4. Instrucţiuni 267
A.2.5. Câteva funcţii utile 269

Capitolul 1

Subprograme

1.1. Noţiunea de subprogram

Definiţia 1.1. Prin **subprogram** vom înţelege un ansamblu alcătuit din declarări şi instrucţiuni scrise în vederea unei anumite prelucrări, ansamblu implementat separat şi identificat printr-un nume.

Până în prezent am fost doar utilizatori de subprograme. Exemple de astfel de subprograme folosite sunt:

- matematice: **`sin`**, **`cos`**, **`abs`**, **`exp`**, **`trunc`** (Pascal) / **`floor`** (C++);
- de manipulare a fişierelor: **`close`** (Pascal) / **`.close()`** (C++).

În acest capitol învăţăm să lucrăm cu subprograme. Pentru a înţelege noţiunea de subprogram, vom porni de la două exemple.

`1.` Se consideră funcţia:

$$f(x)=\begin{cases}\dfrac{x+1}{1+x^2}, & \text{pentru } x\in[-1,1];\\ x+1, & \text{pentru } x\in(-\infty,-1);\\ \dfrac{6}{1+x}, & \text{pentru } x\in(1,\infty).\end{cases}$$

Se citesc două valori reale **`a`** şi **`b`**. Să se scrie un program care afişează care dintre valorile **`f(a)`** şi **`f(b)`** este cea mai mare.

Ce observăm? Că atât pentru calculul valorii funcţiei în punctul **`a`**, cât şi pentru calculul valorii funcţiei în punctul **`b`**, aplicăm acelaşi tip de raţionament: încadrăm valoarea respectivă (**`a`** sau **`b`**) într-unul dintre cele trei intervale şi aplicăm formula de calcul corespunzătoare.

Cum procedăm? Prin utilizarea cunoştinţelor dobândite până în prezent, scriem secvenţa de program care calculează valoarea funcţiei calculată pentru **`x=a`** şi se mai scrie o dată (sau se copiază şi se adaptează) secvenţa de program care calculează valoarea funcţiei calculată pentru **`x=b`**.

Oare nu se poate lucra şi altfel? Răspunsul este afirmativ. Se scrie un subprogram care calculează valoarea funcţiei într-un punct **x** oarecare şi se apelează subprogramul: o dată pentru **x=a** şi încă o dată pentru **x=b**. Valorile calculate la cele două apeluri, se compară între ele şi se afişează rezultatul cerut.

2. Se citeşte **n**, un număr natural. Se citesc apoi **n** numere reale. Se cere să se afişeze cele **n** numere în ordinea crescătoare a valorilor lor.

Desigur, ştim să rezolvăm această problemă în mai multe feluri, pentru că ştim să memorăm un şir de valori (folosind un vector) şi am studiat mai multe metode prin care se poate obţine ordonarea crescătoare a unui şir de valori (folosind algoritmi de sortare). De această dată, vom implementa o metodă cunoscută, cea de sortare, dar vom utiliza subprogramele.

Vom scrie un subprogram care citeşte un vector, unul care tipăreşte un vector şi un al treilea care sortează vectorul după una din metodele cunoscute.

În acest caz, programul ar arăta astfel:

Pasul 1 - se apelează subprogramul care citeşte vectorul.
Pasul 2 - se apelează subprogramul care sortează vectorul.
Pasul 3 - se apelează subprogramul care tipăreşte vectorul.

Faţă de scrierea clasică, aici ***problema a fost descompusă în trei probleme mai simple*** (citire, sortare şi tipărire). În general, *o problemă complexă se rezolvă mai uşor dacă o descompunem în alte subprobleme mai mici.* Apoi, şansele de a greşi la scrierea unui subprogram sunt cu mult mai mici decât acelea de a greşi la scrierea unui program mare.

În plus, dacă într-un alt program este necesară sortarea altui vector de numere reale, metoda clasică ne permite să alegem din secvenţa de instrucţiuni ce formează programul pe cele ce realizează sortarea, să le copiem în noul program şi să facem eventualele adaptări (numărul de componente şi numele vectorului pot fi altele). Aceste operaţii sunt destul de greoaie şi necesită mu ă atenţie. Prin implementarea modulară, cu ajutorul subprogramelor, "preluarea" se realizează mult mai uşor.

Putem acum enumera unele dintre **avantajele** utilizării subprogramelor:

- **reutilizarea codului** - odată scris, un subprogram poate fi utilizat de către mai multe programe;
- **elaborarea algoritmilor prin descompunerea problemei în altele mai simple** - în acest fel, rezolvăm cu mult mai uşor problema;
- **reducerea numărului de erori** care pot apărea la scrierea programelor;
- **depistarea cu uşurinţă a erorilor** - verificăm la început subprogramele, apoi modul în care le-am asamblat (le-am apelat din cadrul programului);
- **realizarea unor programe uşor de urmărit** (lizibile).

1.2. Subprograme în Pascal

1.2.1. Un exemplu de utilizare a funcţiilor

❒ **Problema 1.1.** Se citeşte `n`, număr natural. Să se scrie programele care afişează valoarea calculată a expresiilor:

$$E_1 = 1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}; \quad E_2 = \left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}\right)^n.$$

☑ **Rezolvare.** Funcţia care calculează E_1 este:

```
function subp(n:integer):real;
var s:real;
    i:integer;
begin
 s:=0;
 for i:=1 to n do s:=s+1/i;
 subp:=s;
end;
```

Analiza programului anterior

➔ Antetul funcţiei este "`function subp(n:integer):real;`".

➔ Funcţia se numeşte "`subp`".

➔ Ea este recunoscută de compilator prin faptul că antetul este precedat de cuvântul cheie "`function`".

➔ Funcţia are un parametru numit `n`. Rolul său este important şi anume precizează ***pentru ce valoare trebuie calculată expresia***. Aşa cum vom vedea, există posibilitatea să avem mai mulţi parametri, de diferite tipuri.

➔ Funcţia are un anumit **tip**, care ***precizează natura rezultatului***. În exemplu, tipul este `real` întrucât expresia calculată este de acest tip.

➔ Funcţia are variabile proprii - adică variabile care sunt definite în cadrul ei. În exemplu, ele sunt `s` şi `i`. Aceste variabile se numesc **variabile locale**.

➔ Am văzut că funcţia întoarce un anumit rezultat - în exemplu, de tip `real`. Observaţi mecanismul prin care am obţinut aceasta. Calculez expresia în mod obişnuit. Rezultatul este reţinut de variabila locală `s`. Prin atribuirea "`subp=s;`", funcţia a primit ca valoare de retur conţinutul variabilei `s`.

În continuare, prezentăm cele două programe care utilizează funcţia:

```
var n:integer;
function subp(n:integer):real;
var s:real;
    i:integer;
begin
 s:=0;
 for i:=1 to n do s:=s+1/i;
 subp:=s;
end;
begin
 write ('n='); readln(n);
 write(subp(n):5:2);
end.
```

- - - - - - - - - - - - - - - - - - -

```
var n,i:integer;
    rez,prod:real;
function subp(n:integer):real;
var s:real;
    i:integer;
begin
 s:=0;
 for i:=1 to n do s:=s+1/i;
 subp:=s;
end;
begin
 write ('n='); readln(n);
 rez:=subp(n);
 prod:=1;
 for i:=1 to n do prod:=prod*rez;
 write(prod:5:2);
end.
```

➔ Funcţia este apelată întotdeauna din cadrul unei expresii.

Apelul funcţiei în primul program s-a realizat astfel: "`write(subp(n):5:2);`", iar în al doilea, "`rez:=subp(n)`". În primul caz expresia este trecută ca parametru pentru `write`, iar în al doilea, avem o expresie de atribuire.

În terminologia utilizată în teoria subprogramelor - în particular, în cazul funcţiilor - se utilizează termenii **parametri formali** şi **parametri efectivi**.

Definiţia 1.2. Parametrii care se găsesc în antetul funcţiei se numesc **parametri formali**.

Atunci când scriem o funcţie, nu cunoaştem valoarea propriu-zisă a parametrilor. Funcţia trebuie să întoarcă rezultatul corect, oricare ar fi valoarea lor. Din acest punct de vedere ei se numesc **formali**.

Definiţia 1.3. Parametrii care se utilizează la apel se numesc **parametri efectivi**.

La apel, lucrurile stau altfel: valorile acestora sunt cunoscute. Prin urmare aceştia se numesc parametri **efectivi**.

Între parametrii formali si parametrii efectivi trebuie să existe o concordanţă, care va fi studiată la momentul potrivit.

1.2.2. Un exemplu de utilizare a procedurilor

❒ **Problema 1.2.** Se citeşte **n**, număr natural. Se citeşte un vector cu **n** componente numere reale. Se cere să se tipărească vectorul sortat.

☑ **Rezolvare.** Programul este următorul:

```
type vector=array [1..9] of integer;
var  n,i:integer;
     v:vector;

procedure citesc;
begin
 write('n='); readln(n);
 for i:=1 to n do
  begin
   write('v[',i,']='); readln(v[i]);
  end;
end;

procedure tiparesc;
begin
 for i:=1 to n do writeln(v[i]);
end;

procedure sortez;
var  gasit:boolean;
     man:integer;
begin
 repeat
  gasit:=false;
  for i:=1 to n-1 do
   if v[i]>v[i+1]
    then
     begin
      gasit:=true;
      man:=v[i]; v[i]:=v[i+1]; v[i+1]:=man
     end
  until not gasit;
end;

begin
 citesc;
 sortez;
 tiparesc;
end.
```

Programul conţine trei proceduri: **citesc**, **sortez** şi **tiparesc**. Forma simplificată prin care este descrisă o procedură este:

```
procedure nume;
begin
.........
end;
```

- ***Apelul** unei proceduri se face prin **utilizarea numelui ei**, sub forma* `nume;`. Mai precis, *apelul unei proceduri este instrucţiune*, numită **instrucţiunea de apel**.
- *La apel, controlul programului este transferat la prima instrucţiune a procedurii* - după cum se întâmplă şi la funcţii. După executarea procedurii, *se revine în programul principal la prima instrucţiune care urmează celei de apel* - în cazul de faţă se întâlneşte o altă instrucţiune de apel.
- Ca şi funcţiile, *procedurile **se pot plasa în cadrul programului** între declaraţiile de variabile şi instrucţiunea compusă.*
- *Orice variabilă a programului principal este şi variabilă a procedurii (invers nu este adevărat).*

În acest exemplu, procedurile sunt apelate fără utilizarea parametrilor. În realitate, procedurile pot avea parametri întocmai ca şi funcţiile.

1.2.3. Structura unui subprogram

În esenţă, un subprogram este alcătuit din:

- **Antet** - conţine mai multe informaţii importante necesare compilatorului, *numele subprogramului, lista parametrilor formali.* În cazul subprogramelor de tip funcţie, conţine şi tipul rezultatului (valoarea întoarsă de funcţie).
- **Bloc** - cuprinde definiţiile tipurilor de variabile, ale variabilelor, ale subprogramelor şi o instrucţiune compusă. Structura sa este identică cu cea a programului principal, indiferent dacă este bloc de procedură sau de funcţie.

Programul principal conţine un antet (program nume) şi un bloc.

Pentru prezentarea structurii subprogramelor de tip funcţie vom utiliza diagramele de sintaxă.

1.2.3.1. Structura subprogramelor de tip funcţie

În figura următoare este prezentată **structura antetului**:

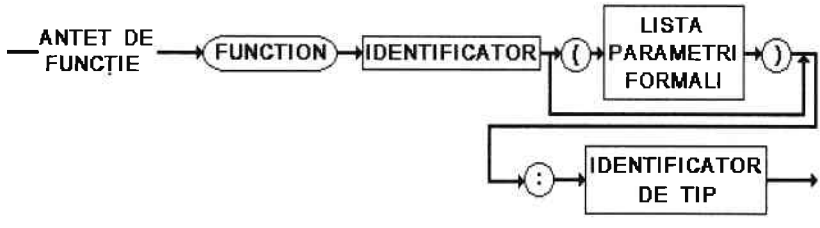

Figura 1.1. Structura antetului unui subprogram de tip funcţie

Exemplu:

```
function suma(x,y:integer):real;
 begin
  suma:=x+y;
 end
```

- antetul este "`function suma(x,y:integer):real;`";

- identificatorul (numele) funcţiei este **`suma`**. Lista parametrilor formali este: "`x,y:integer`". Funcţia este de tip **`real`** (identificatorul de tip);

- blocul este (în acest caz lipsesc variabilele locale):

```
begin
  suma:=x+y;
end
```

Important: identificatorul de tip (tipul valorii întoarse de funcţie) poate fi: de orice tip ordinal, de orice tip real, de orice tip pointer sau de tipul **`string`** (acest tip va fi studiat în acest manual).

1.2.3.2. Structura subprogramelor de tip procedură

În figura următoare este prezentat **antetul de procedură**:

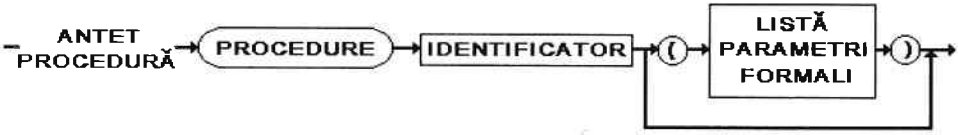

Figura 1.2. Structura antetului unui subprogram de tip procedură

Exemplu:

```
procedure suma(var s:real; x,y:real);
begin
 s:=x+y;
end;
```

- antetul este: "`procedure suma(var s:real; x,y:real);`"

- identificatorul (numele procedurii) este **`suma`**;

- lista parametrilor formali este: "`var s:real; x,y:real;`";

- blocul este:

```
begin
 s:=x+y;
end;
```

1.2.4. Definirea şi declararea unui subprogram

Deşi aparent asemănătoare, cele două noţiuni diferă. Buna înţelegere a lor ne ajută să evităm anumite erori. Mai mult, aceste noţiuni sunt utilizate de orice limbaj de programare, nu numai de **Pascal**.

Definiţia 1.4. A defini un subprogram, înseamnă a-l scrie efectiv, după structura anterior prezentată. O problemă importantă este locul unde se defineşte subprogramul.

În Pascal, subprogramele se definesc în trei locuri:

1. ***În blocul programului care-l utilizează*** - numit şi **program principal** - între declaraţia variabilelor şi instrucţiunea compusă:

```
program exemplu1;
var x:integer;
procedure t;
begin
 writeln(x)
end;
begin
 x:=3;
 t
end.
```

2. ***În cadrul unităţilor de program*** - vor fi studiate ulterior.

3. ***În blocul unui alt subprogram***. Această formă este proprie limbajului Pascal. În exemplu, procedura `t`, are ca subprogram procedura `z`. Iniţial programul va atribui variabilei `x` valoarea 3. Apoi va apela procedura `t`. Ea tipăreşte conţinutul lui `x` (3), apoi apelează procedura `z`. La rândul ei, aceasta afişează un mesaj: `'eu sunt z'`.

```
program exemplu2;
var x:integer;

procedure t;
  procedure z;
   begin
    writeln('eu sunt z');
   end;
 begin
  writeln(x); z;
 end;

begin
 x:=3;
 t
end.
```

Definiţia 1.5. A declara un subprogram, înseamnă a-l anunţa. Un subprogram nedeclarat nu poate fi folosit.

1. *În cazul în care subprogramul este* ***definit în blocul programului principal sau în blocul subprogramului****, se consideră că a fost şi declarat pentru acesta, deci poate fi folosit.* În acest caz, declaraţia coincide cu definiţia.

Exemple

- În programul **exemplu1**, procedura **t** este definită în blocul programului principal. Ea poate fi apelată din programul principal.

- În programul **exemplu2**, procedura **t** este definită în blocul programului principal, deci poate fi apelată de acesta. Procedura **z** este definită în blocul procedurii **t**. Ea poate fi apelată din **t**. În schimb, **nu** poate fi apelată din programul principal.

2. În blocul programului principal sau în blocul unui subprogram ***se definesc, în ordine, subprogramele*** s_1, s_2, ..., s_k. În acest caz, automat s_1 este declarat pentru s_2, s_3, ..., s_k, apoi s_2 este declarat pentru s_3, ..., s_k, iar s_{k-1} este declarat pentru s_k.

Exemple

- Procedurile **s1** şi **s2** pot fi apelate din programul principal. Procedura **s1** poate fi apelată din procedura **s2**, dar **s2** nu poate fi apelată, în absenţa altei clauze, din **s1**:

```
procedure s1;
 begin
 writeln('s1')
 end;

procedure s2;
 begin
  s1;
  writeln ('s2');
 end;

begin
 s1;
 s2;
end.
```

- Procedura **test** conţine definiţia a două proceduri **s1** şi **s2**. Nici una din aceste proceduri nu se consideră declarată pentru programul principal. Amândouă sunt declarate pentru procedura **test**. Pentru procedura **s2**, **s1** este declarată, dar pentru **s1** procedura **s2** nu este declarată.