

SUBPROGRAME

SUBPROGRAMUL reprezinta parti identificabile prin nume care se pot activa la cerere prin intermediul aceluiași nume. O parte din subprogram se contruiește ca subprogram dacă un algoritm cuprinde în mai multe locuri aceeași secvență de operații executabile pentru aceleași date sau pentru date diferite. În loc ca subprogramul să cuprindă în același loc, același grup de instrucțiuni, concepe grupul de instrucțiuni ca subprogram, el va apărea în program o singură dată și se va activa de mai multe ori. Partea respectivă de program rezolvă o subproblemă din cele în care se descompune problema complexă. În limbajul Pascal, avem două tipuri de subprograme : procedurile și funcțiile. Deosebirea între ele constă în numărul de valori calculate și returnate programului apelat. Procedurile calculează mai multe valori sau nici una, iar funcțiile returnează o singură valoare asociată numelui funcției. Atât procedurile cât și funcțiile pot fi standard (predefinite în unitul sistem), cât și nestandard (definite de utilizator). Procedurile și funcțiile nestandard trebuie declarate obligatoriu înainte de a fi apelate.

O declarație de subprograme cuprinde:

- un antet de subprogram care precizează interfața subprogramului cu mediul sau, și
- blocul subprogramului care descrie funcționarea lui internă.

DOMENIUL DE VIZIBILITATE AL IDENTIFICATORILOR

Prin domeniul de vizibilitate (valabilitate) se înțelege zona de program în care e valabilă declararea sau definirea unui identificator. Toți identicatorii definiți sau declarați într-un bloc sunt cunoscuți în blocul respectiv și se numesc variabile locale. Dacă blocul cuprinde blocuri incluse în care identicatorii (variabile locale ale acestora) nu se definesc sau redenumesc, atunci acestea sunt cunoscute în blocurile incluse și se numesc variabile globale pentru acesta. Dacă o variabilă declarată într-un bloc se redefineste atunci în blocul în care a fost redeclarată va fi variabilă atribuită generată la redeclarare.

DECLARAREA ȘI APELUL PROCEDURILOR. PARAMETRII FORMALI ȘI PARAMETRII EFECTIVI

O procedură e un subprogram care calculează mai multe valori accesibile sau nu programului apelant sau efectuează anumite operații fără să calculeze vreă valoare. Valorile calculate accesibile programului apelant reprezintă parametrii de ieșire ai subprogramului. Aceștia pot depinde de anumite valori pe care subprogramul le primește

din programul apelant, valori reprezentand parametrii de intrare. Parametrii formali sunt variabile simbolice in care lucreaza subprogramul. Ele sunt declarate in antetul subprogramului si sunt cunoscute numai in interiorul subprogramului. La apelarea procedurii se specifica parametrii efectivi sau actuali prin intermediul instructiunii procedurale. Parametrii efectivi reprezinta variabilele cu care subprogramele lucreaza efectiv in momentul activarii.

Declararea procedurii se face folosind:

PROCEDURE nume_procedura(lista parametrii)

-parametrii precizati la scrierea procedurii sunt parametrii formali si se separa prin ‘;’,

-pentru fiecare parametru se precizeaza numele si tipul acestuia.

Apelarea procedurii :

Pentru a executa o procedura aceasta trebuie apelata. La apel se da numele procedurii si valorile concrete ale parametrilor care se separa prin punct si virgula.

Ex : procedure citire(n :integer ; k :char) ;

Begin

.....

end;

Cand se apeleaza o procedura, modulul apelant a abandonat temporar, si se executa procedura. In timpul executiei procedurii, parametrii formali sunt inlocuiti in tot corpul procedurii cu parametrii actuali (valori concrete). Dupa executarea procedurii se revine in modulul apelant la linia imediat urmatoare celei care a facut apelul. Parametrii formali si parametrii efectivi nu e obligatoriu sa aiba acelasi nume dar trebuie sa existe o concordanta de numar, tip si ordine.

DECLARAREA SI APELUL FUNCTIILOR

O functie e un subprogram care calculeaza si returneaza programului apelant o singula valoare. Aceasta valoare este asociata numelui functiei. Iar tipul poate fi simplu, string sau reper. Valoarea returnata de functie nu poate avea alt tip structurat decat string.

Declararea unei functii:

FUNCTION nume_functie(lista parametrii formali): identificador de tip;

-nume_functie reprezinta numele functiei, al carei tip este ‘identificador de tip’

-identificador de tip = nume de tip simplu: STRING sau REPER;

Blocul functiei trebuie sa contina obligatoriu o instructiune de atribuire prin care identificadorul functiei primeste valoarea unei expresii.

Identificadorul functiei nu are voie sa apara in partea dreapta a unor atribuirii decat daca functia este recursiva.

Apelul unei functii decurge astfel:

- se intrerupe calculul expresiei in care a aparut apelul functiei ;

- se transmit parametrii, daca exista, exact ca la proceduri ;

- se executa functia;

METODA BACKTRACKING

Se aplica problemelor in care solutia poate fi reprezentata sub forma unui vector – $x=(x_1, x_2, x_3, \dots, x_k, \dots, x_n) \in S$, unde S este multimea solutiilor problemei si $S=S_1 \times S_2 \times \dots \times S_n$, si S_i sunt multimi finite avand s_i elemente si $x_i \in S_i, (\forall) i = 1..n$.

Pentru fiecare problema se dau relatii intre componentele vectorului x , care sunt numite conditii interne ; solutiile posibile care satisfac conditiile interne se numesc solutii rezultat. Metoda de generare a tuturor solutiilor posibile si apoi de determinare a solutiilor rezultat prin verificarea indeplinirii conditiilor interne necesita foarte mult timp.

Metoda backtracking evita aceasta generare si este mai eficienta. Elementele vectorului x , primesc pe rand valori in ordinea crescatoare a indicilor, $x[k]$ va primi o valoare numai daca au fost atribuite valori elementelor $x_1.. x[k-1]$. La atribuirea valorii lui $x[k]$ se verifica indeplinirea unor conditii de continuare referitoare la $x_1.. x[k-1]$. Daca aceste conditii nu sunt indeplinite, la pasul k , acest lucru inseamna ca orice valori i am atribui lui $x[k+1], x[k+1], .. x[n]$ nu se va ajunge la o solutie rezultat.

Metoda backtracking construiesc un vector solutie in mod progresiv incepand cu prima componenta a vectorului si mergand spre ultima cu eventuale reveniri asupra atribuirilor anterioare.

Metoda se aplica astfel :

- 1) se alege prima valoare din S_1 si x_1 se atribuie lui x_1 ;
- 2) se presupun generate elementele $x_1..x[k-1]$, cu valori din $S_1..S[k-1]$; pentru generarea lui $x[k]$ se alege primul element din $S[k]$ disponibil si pentru valoarea aleasa se testeaza indeplinirea conditiilor de continuare.

Pot aparea urmatoarele situatii :

- a) $x[k]$ indeplineste conditiile de continuare. Daca s-a ajuns la solutia finala ($k=n$) atunci se afiseaza solutia obtinuta. Daca nu s-a ajuns la solutia finala se trece la generarea elementului urmator – $x[k-1]$;
- b) $x[k]$ nu indeplineste conditiile de continuare. Se incearca urmatoarea valoare disponibila din $S[k]$. Daca nu se gaseste nici o valoare in $S[k]$ care sa indeplineasca conditiile de continuare, se revine la elementul $x[k-1]$ si se reia algoritmul pentru o noua valoare a acestuia. Algoritmul se incheie cand au fost luate in considerare toate elementele lui S_1 .

Problemele rezolvate prin aceasta metoda necesita timp mare de executie, de aceea este indicat sa se foloseasca metoda numai daca nu avem alt algoritm de rezolvare.

Daca multimile S_1, S_2, \dots, S_n au acelasi numar k de elemente, timpul necesar de executie al algoritmului este k la n . Daca multimile $S_1, S_2.. S_n$ nu au acelasi numar de elemente, atunci se noteaza cu ‘ m ’ minimul cardinalelor multimilor $S_1..S_n$ si cu ‘ M ’, maximul. Timpul de executie este situat in intervalul $[m \text{ la } n .. M \text{ la } n]$. metoda backtracking are complexitatea exponentiala, in cele mai multe cazuri fiind inefficienta. Ea insa nu poate fi inlocuita cu alte variante de rzelvare mai rapide in situatia in care se cere determinarea tuturor solutiilor unei probleme.

RECURSIVITATE

Prin recursivitate se intelege faptul ca un subprogram se apeleaza pe el insusi, apelul aparand atunci cand subprogramul este inca activ. Exista doua tipuri de recursivitate:

- 1) recursivitate directa – cand un subprogram se autoapeleaza in corpul sau ;
- 2) recursivitate indirecta – cand avem doua subprograme (x si y), iar x face appel la y si invers ;

Se folosesc algoritmi recursivi atunci cand calculele aferente sunt descrise in forma recursiva.

Recursivitatea este frecvent folosita in prelucrarea structurilor de date definite recursiv. Un subprogram recursiv trebuie scris astfel incat sa respecte regulile :

- a) Subprogramul trebuie sa poata fi executat cel putin o data fara a se autoapela ;
- b) Subprogramul recursiv se va autoapela intr-un mod in care se tinde spre ajungerea in situatia de executie fara autoapel.

Pentru a permite apelarea recursiva a subprogramelor, limbajul Pascal dispune de mecanisme speciale de suspendare a executiei programului apelant, de salvare a informatiei necesare si de reactivare a programului suspendat .

Pentru implementarea recursivitatii se foloseste o zona de memorie in care se poate face salvarea temporala a unor valori. La fiecare appel recursiv al unui subprogram se salveaza in aceasta zona de memorie starea curenta a executiei sale.

Deși variabilele locale ale subprogramului apelant au aceleasi nume cu cele ale subprogramului apelat, orice referire la acesti identificatori se asociaza ultimului set de valori alocate in zona de memorie. Zona de memorie ramane alocata pe tot parcursul executie subprogramului apelat si se dealoca in momentul revenirii in programul apelat. Zona de memorie nu este gestionata explicit de programator ci de catre limbaj.

La terminarea executiei subprogramului apelat recursiv, se reface contextul programului din care s-a facut apelul. Datorita faptului ca la fiecare autoapel se ocupa o zona de memorie, recursivitatea este eficienta numai daca numarul de autoapelari nu este prea mare pentru a nu se ajunge la umplerea zonei de memorie alocata.

Recursivitatea ofera avantajul unor solutii mai clare pentru probleme si a unei lungimi mai mici a programului. Ea prezinta insa dezavantajul unui timp mai mare de executie si a unui spatiu de memorie alocata mai mare. Este de preferat ca atunci cand programul recursiv poate fi transformat intr-unul iterativ sa se faca apel la cel din urma.