Metode de proiectarea algoritmilor - Proiectarea modulara

Prin programare modulara, un algoritm se va rezolva prin folosirea modulelor. Modulul este considerat o unitate de sine statatoare, care poate fi un program, un subprogram sau o unitate de program.

Programarea modulara se bazeaza pe descompunerea problemei In subprograme şi proiectarea şi programarea separata a subalgoritmilor corespunzatori.

Un **subprogram** este o colecție de tipuri de date, variabile, instrucțiuni care indeplinesc o anumită sarcină (calcule, citiri, afișări), atunci când este apelat (folosit) de un program sau de un alt subprogram.
Avantaje utilizare subprograme:

- *reutilizarea codului* – după ce am scris un subprogram il putem apela de oricâte ori este nevoie;
- *modularizarea programelor* – subprogramele ne permit să impărțim problema dată in mai multe subprobleme, mai simple;
- *reducerea numărului de erori* care pot să apară in scrierea unui program;
- *depistarea cu ușurință a erorilor* – fiecare subprogram va fi verificat la crearea sa, apoi verificăm modul in care apelăm subprogramele.

Tipuri de subprograme:

1. ***funcții*** – subprograme care determină un anumit rezultat, o anumită valoare, pornind de la anumite date de intrare. Spunem că valoarea este returnată de către funcție, iar aceasta va fi apelată ca operand intr-o expresie, valoarea operandului in expresie fiind de fapt valoarea rezultatului funcției.
2. ***proceduri*** – subprograme care se folosesc intr-o instrucțiune de sine stătătoare, nu intr-o expresie. Ele indeplinesc o sarcină, au un efect și nu returnează un rezultat. De exemplu, citirea unor variabile, afișarea unor valori, transformarea unor date, etc. .

Subprogramele sunt părţi ale unui program, identificabile prin nume, care se pot activa la cerere prin intermediul acestor nume.

Utilizarea subprogramelor intr-un program presupune abordarea a două noţiuni: **definirea** unui subprogram şi **apelul** unui subprogram.

Definirea unui subprogram reprezintă de fapt descrierea unui proces de calcul cu ajutorul variabilelor virtuale (**parametri formali**) iar apelul unui subprogram este execuţia procesului de calcul pentru cazuri concrete (cu ajutorul **parametrilor reali**,

(efectivi, actuali)). Definirea unui subprogram presupune declararea antetului și a corpului.

Ex.:

tip_returnat *nume_funcţie* (lista parametrilor formali) //antet funcție
{
instrucţiune; // corpul funcţiei
}

*tip_returnat – repr. tipul rezultatului calculat si returnat de functie (int, char, long, float etc.). Daca tipul rezultatului este diferit de void, corpul functiei trebuie sa contina cel putin o instructiune return. Aceasta va specifica valoarea calculata si returnata de functie, care totodata trebuie sa fie de acelasi tip ca si tip_returnat;
*nume_functie – numele dat subprogramului (identificator);
*lista parametrilor formali – lista de declaratii de variabile separate prin virgula (poate fi si vida);
Atunci când se execută o funcţie, ea foloseşte parametrii actuali care i-au fost transmişi prin apelul său, ţinând cont de natura parametrilor formali.

In cazul **transmiterii parametrilor prin valoare**, parametrii formali ai unei funcții sunt copii ale valorilor parametrilor actuali. Acest lucru inseamnă că:
- parametri actuali pot fi expresii ale căror valori corespund ca tip cu parametri formali (sau pot fi convertite implicit la tipul parametrilor formali);
- pe stivă se memorează valoare expresiei (sau variabilei) date ca parametru actual;
- la ieșirea din apelul funcției modificările realizate in funcție asupra parametrilor formali nu au efect asupra parametrilor actuali. Parametrii actuali sunt nemodificați!
- acesta este modul implicit de transmitere a parametrilor.

Transmiterea prin referinta este mecanismul prin care putem modifica intr-o funcție variabile din afara funcției. In cazul transmiterii parametrilor prin referință, parametrii formali ai unei funcții sunt referințe ale parametrilor actuali. Acest lucru inseamnă că:
- parametri actuali pot fi doar variabile, sau expresii ale căror rezultate sunt similare variabilelor: elemente de tablou, câmp al unei structuri, pointer dereferențiat, etc.;
- pe stivă se memorează adresa variabilei date ca parametru actual;
- toate modificările realizate in apelul funcției asupra parametrilor formali se fac de fapt asupra parametrilor actuali. Parametrii actuali sunt modificați la ieșirea din apel!
- pentru a preciza că un parametru este transmis prin referință va fi precedat de caracterul & in antetul funcției.

Ex.:

```
<?php
function add_some_extra(&$string)
{
$string .= 'dar si 3 pere.';
```

```
}
$str = 'Ana are 2 mere, ';
add_some_extra($str);
echo $str;
?>
```

- `rezultat: 'Ana are 2 mere, dar si 3 pere.'`

Apelul subprogramului este modul prin care subprogramul este pus in execuţie. Apelul subprogramului se poate realiza in două moduri:

- printr-o instructiune de apel;
- ca operand intr-o expresie.

Instructiunea de apel a unui subprogram are urmatorul format general:

nume_functie (lista_parametrilor_actuali);

Se utilizează instrucţiuni de apel atunci când subprogramul nu returnează nici o valoare sau când nu se doreşte utilizarea valorii returnate de subprogram, ci doar efectuarea prelucrărilor descrise de subprogram.

In cazul in care se doreşte utilizarea valorii returnate de subprogram ca operand intr-o expresie, se va apela subprogramul in cadrul expresiei astfel:

nume_functie (lista_parametrilor_actuali)

In această situaţie lipseşte caracterul ‘;’, care marchează sfârşitul instrucţiunii de apel.

La apelul unui subprogram, valorile parametrilor actuali sunt atribuite, in ordine, parametrilor formali corespunzători. Atât procedurile, cât şi funcţiile, trebuie definite inainte de a fi apelate. Apelarea unei funcţii nu este o instrucţiune de sine stătătoare , ea trebuie inclusă ca operant in cadrul unei expresii.

Apelul unui subprogram provoacă o serie de operații secundare, pe care programatorul nu le vede direct, dar de care trebuie să țină seama. Reguli de lucru cu variabile de lucru: - identificatorii declarați in cadrul unui subprogram sunt vizibili numai in interiorul acelui subprogram și se numesc elemente locale sau automate, fiind automat create când subprogramul este apelat și depuse pe stiva sistemului;

- identificatorii care desemnează parametrii unui subprogram au statut de variabile locale;

- elementele declarate in afara modulelor au statut de elemente globale și sunt cunoscute in tot fișierul text al programului.

O funcţie recursivă este o funcţie care se apelează pe ea insăşi. Există două tipuri de funcţii recursive:

- funcţii cu un singur apel recursiv, ca ultimă instrucţiune, care se pot rescrie sub formă nerecursivă (iterativă);
- funcţii cu unul sau mai multe apeluri recursive, a căror formă trebuie să folosească o stivă pentru memorarea unor rezultate intermediare.

Recursivitatea este posibilă deoarece, la fiecare apel al funcţiei, adresa de revenire, variabilele locale şi parametrii formali sunt memorate intr-o stivă, iar la ieşirea din funcţie, se scot din stivă toate datele puse la intrarea in funcţie.

O funcţie recursivă trebuie să conţină cel puţin o instrucţiune *if*, de obicei la inceput, prin care se verifică dacă este necesar un apel recursiv sau se iese din funcţie.
Variabilă locală: o variabilă locală este o variabilă care este declarată in structura principală a unei metode și este limitată la domeniul de aplicare local pe care i se dă. Variabila locală poate fi utilizată numai in metoda in care este definită și, dacă ar fi folosită in afara metodei definite, codul va inceta să funcționeze.
Variabilă globală: o variabilă globală este o variabilă care este declarată in afara oricărei alte funcții definite in cod. Datorită acestui fapt, variabilele globale pot fi utilizate in toate funcțiile, spre deosebire de o variabilă locală.
Observatii:

- variabilele globale pot fi modificate in interiorul subprogramelor, dar trebuie acordată mare atenţie prelucrării acestora;
- utilizarea variabilelor globale trebuie să fie limitată deoarece creează dependenţe intre subprograme;
- utilizarea variabilelor locale asigură portabilitatea programelor;
- utilizarea constantelor globale permite modificarea, la nevoie, intr-un singur loc a unei valori ce afectează intregul program şi subprogramele sale;
- un program care are mai mulţi parametri, adică foloseşte mai puţine variabile globale, este mai flexibil şi mai portabil;
- un program care are mai mulţi parametri, adică foloseşte mai puţine variabile globale, este mai uşor de inţeles, de depanat şi de apelat;

In cazul in care există o variabilă locală care are acelaşi nume cu al unei variabile globale, aceste două variabile se numesc variabile omonime. Variabilele locale sunt prioritare variabilelor globale omonime.

A defini un subprogram inseamnă a-l scrie efectiv, după o anumită structură. A declara un subprogram inseamnă a-l anunţa. Un subprogram nedeclarat nu poate fi folosit. Definiţia unui subprogram ţine loc şi de declaraţie.
Orice program trebuie să conţină :

- instrucţiuni imperative, prin care se comandă executarea anumitor acţiuni;
- declaraţii de variabile, de funcţii, etc. necesare compilatorului, dar fără efect la execuţie;
- comentarii, ignorate de compilator, necesare utilizatorului.

Motivele utilizarii de subprograme sunt multiple:

- un program mare poate fi mai uşor de scris, de inţeles şi de modificat daca este modular, deci format din module funcţionale relativ mici.
- un subprogram poate fi reutilizat in mai multe aplicaţii, ceea ce reduce efortul de programare al unei noi aplicaţii.
- un subprogram poate fi scris şi verificat separat de restul aplicaţiei, ceea ce reduce timpul de punere la punct a unei aplicaţii mari (deoarece erorile pot apare numai la comunicarea intre subprograme corecte).

- intreţinerea unei aplicaţii este simplificata, deoarece modificarile se fac numai in anumite subprograme şi nu afecteaza alte subprograme (care nici nu mai trebuie recompilate).

Prin algoritm se intelege o metoda de solutionare a unei clase de probleme, reprezentata de o succesiune finite de operatii bine definite, numite instructiuni.
Caracteristicile unui algoritm sunt:

- este descris clar, fara ambiguitati in privinta ordinii de executare a instructiunilor;
- este corect, deci este o metoda care rezolva problema pe orice caz (rezolva o clasa de probleme);
- este finit, deci se termina dupa un nr. finit de pasi, indiferent cat de multi;
- este realizabil cu resursele disponibile.

Programarea structurată are la bază teorema de structură care afirmă că orice algoritm cu o singură intrare şi o singură ieşire poate fi reprezentat ca o combinaţie de trei tipuri de structuri de control – secvenţa, decizia şi ciclul cu test iniţial. Se mai admite folosirea a încă trei tipuri de structuri de control – selecţia, ciclul cu test final şi ciclul cu contor.

1. Structura liniară (secvenţa) este o succesiune de operaţii ce realizează o prelucrare (transformare) a datelor. Operaţiile sunt executate una după alta, în ordinea scrierii .
Atribuirea: variabilă←expresie;
Operaţia de citire (intrare): citeşte variabila1, variabila2, ... , variabila n;
Operaţia de scriere (ieşire): scrie expresie1, expresie2, ... , expresie n;

2.Structura alternative (decizia) permite alegerea unei operaţii/secvenţe de operaţii din două alternative posibile:
if conditie (C)
then instructiuneA;
else instructiuneB;

Implementarea structurii decizionale:

a. Instructiunea ***if***
   - Sintaxa:
     if (expresie)
       instructiune 1;
     [else instructiune 2;] -- ramura else este optionala.

La intalnirea instructiunii “if”, se evalueaza expresia (conditia) din paranteze. Daca valoarea expresiei este 1 sau diferita de 0 (conditia este indeplinita) se executa instructiune1; daca valoarea expresiei este 0 (conditia nu este indeplinita), se executa instructiune2. Deci, la un moment dat, se executa doar una dintre cele doua instructiuni: fie instructiune1, fie instructiune2. Dupa executia instructiunii “if” se trece la executia instructiunii care urmeaza acesteia.
Observaţii:

1. Instructiune1 si instructiune2 pot fi instructiuni compuse (blocuri), sau chiar alte instructiuni if (if-uri imbricate);
2. Deoarece instructiunea if testeaza valoarea numerica a expresiei (conditiei), este posibila prescurtarea: if (expresie), in loc de if (expresie != 0);
3. Deoarece ramura else a instructiunii if este optionala, in cazul in care aceasta este omisa din secventele if-else imbricate, se produce o ambiguitate. De obicei, ramura else se asociaza ultimei instructiuni if.

b. Instructiunea ***switch***
   - Sintaxa:
     switch (expresie)
     {
     case expresie_const_1: instructiune_1;
     [break;]
     case expresie_const_2: instructiune_2;
     [break;]
     .....
     case expresie_const_n-1: instructiune_n-1;
     [break;]
     [default: instructiune_n;]
     }

Este evaluata expresia (expresie aritmetica), iar valoarea ei este comparata cu valoarea expresiilor constante 1, 2, etc. (expresii constante=expresii care nu contin variabile). In situatia in care valoarea expresie este egala cu valoarea expr_const_k, se executa instructiunea corespunzatoare acelei ramuri (instructiune_k). Daca se intalneste instructiunea break, parcurgerea este intrerupta, deci se va trece la executia primei instructiuni de dupa switch. Daca nu este intalnita instructiunea break, parcurgerea continua. Break-ul cauzeaza deci, iesirea imediata din switch. In cazul in care valoarea expresiei nu este gasita printre valorile expresiilor constante, se executa cazul marcat cu eticheta default (cand acesta exista). Expresiile expresie, expresie_const_1, expresie_const_2,etc., trebuie sa fie intregi.

Operatorul de atribuire (=) este un operator binar, care permite modificarea valorii unei variabile. Forma generală a unei operaţii de atribuire este:

variabila = expresie;

Efect:

- se calculează valoarea expresiei şi se obţine un rezultat;
- rezultatul se memorează la adresa variabilei;
- efectul acestei operaţii este întotdeauna de la dreapta la stânga.

Exista mai multe metode de interschimbare a valorilor dintre doua variabile:

1. regula celor 3 pahare – implica folosirea unei variabile auxiliare;
   citeste a

citeste b
scrie a, b (initial)
c=a
a=b
b=c
scrie a, b (final)

2. prin adunari si scaderi:
   citeste a
   citeste b
   scrie a, b (initial)
   a = a + b
   b = a – b
   a = a – b
   scrie a, b (final)
3. prin inmultiri si impartiri:
   citeste a
   citeste b
   scrie a, b (initial)
   a = a*b
   b = a/b
   a = a/b
   scrie a, b(final)

Exemple aplicatii:

1. maximul a 3 numere
   citeste a
   citeste b
   citeste c
   daca a > b atunci
   daca a > c atunci
   max = c
   altfel
   max = a
   altfel
   daca c > b atunci
   max = c
   altfel
   max = b
   scrie max

2. ecuatie de gradul 1: ax + b = 0

```
reale a,b
citeste a
citeste b
daca a=0 atunci
        daca b=0 atunci
                        scrie “infinitate de solutii”
                altfel
                        scrie “imposibil”
        altfel x = -b/a
scrie x
```