NOŢIUNI INTRODUCTIVE DE C++

1 Introducere

C++ este un limbaj de programare care a apărut prin extinderea limbajului C la începutul anilor '80 de către Bjarne Stroustrup şi care, pe lângă posibilitatea programării structurate, oferă şi pe cea a programării orientată pe obiecte. În cele ce urmează ne vom concentra atenţia asupra programării structurate şi vom realiza o rapidă trecere în revistă a principalelor elemente ale limbajului.

Obiectivul nostru este acela de a crea programe care să aproximeze soluţiile diferitelor probleme matematice şi care să fie corecte, eficiente, fiabile, generale şi cât mai elegante.

Realizarea unui program are mai multe etape, între care:

- **Editarea** permite crearea sau modificarea textului sursă (program). De regulă, introducerea sau schimbarea programului se realizează cu ajutorul editorului integrat. Periodic se va realiza salvarea datelor introduse prin apăsarea tastei F2. Textele sursă se păstrează în fişiere cu extensia .CPP.

  Pentru scrierea unui nou program se poate folosi fereastra NONAME00.CPP care se deschide odată cu lansarea programului sau prin secvenţa File/New (dacă extensia nu este .CPP, aceasta va trebui schimbată imediat). Pentru modificarea unui program deja existent se încarcă acesta prin secvenţa File/Open şi apoi prin selectarea, în fereastra deschisă, a directorului unde se află şi a numelui său .

- **Compilarea** traduce textul sursă (programul) din fereastra activă de editare în aşa-numitul cod-maşină. În acest fel se obţine un program-maşină care este păstrat în memorie într-un fişier .EXE pentru o folosire ulterioară.

  Compilarea se realizează prin secvenţa Compile/Compile sau prin acţionarea simultană a tastelor Alt+F9. Dacă la compilare se găsesc erori (care provin din folosirea greşită a elementelor de limbaj) se va afişa o listă de erori de compilare în care se specifică tipul erorii şi localizarea ei.

- **Execuţia programului** permite obţinerea rezultatelor dorite. Ea se realizează prin secvenţa Run/Run sau prin acţionarea simultană a tastelor Ctrl+F9.

  În mod comun, datele obţinute sunt afişate pe ecranul utilizatorului, în timp ce, la terminarea execuţiei, controlul este redat mediului C++. Vizualizarea rezultatelor programului se realizează prin secvenţa Debug/User screen sau prin acţionarea simultană a tastelor Alt+F5. În loc de afişarea lor pe ecranul utilizatorului, este posibilă păstrarea rezultatelor într-un fişier. În acest fel se permite utilizarea şi prelucrarea lor ulterioară (realizarea de grafice, tabele comparative, etc).

2 Directivele preprocesorului

La începutul oricărui program sunt incluse o serie de directive care sunt executate înainted de compilare (faza de preprocesare). Ele nu sunt instrucţiuni şi prin urmare nu sunt urmate de

";". Cele mai fecvente directive sunt:

- Includerea fişierelor antet prin directiva

  #include

  Aceasta este urmată de numele fişierului antet utilizat de program între paranteze unghiulare < >. Aceste fişiere au extensia .h şi conţin definiţii şi instrucţiuni necesare pentru realizarea multor funcţii utile precum: introducerea şi afişarea datelor (iostream.h), folosirea unor funcţii matematice (math.h), citirea sau scrierea datelor în fişiere de pe disc (fstream.h), convertirea numerelor în text şi invers, funcţia exit (stdlib.h), manipularea timpului şi a datei (time.h), manipularea ecranului (conio.h) etc.

- Definirea unor constante simbolice sau macro-uri prin directiva :

  #define

  Se asociază unui simbol o valoare constantă (care nu se modifică pe parcursul programului), de exemplu

  #**define** *max 100*

  sau unui identificator cu sau fără argumente o expresie care depinde de acestea, precum

  #**define** *aria(x,y) ((x)*(y))*

  Definirea unor constante simbolice este foarte utilă atunci când se urmăreţe schimbarea rapidă a unei mărimi care apare în multe locuri în cuprinsul programului şi când este necesară cunoaşterea valorii unei variabile la compilare. Definirea unor macro-uri uşurează citirea şi scrierea programelor.

3 Tipuri de date şi variabile

Principalele tipuri de date pe care le vom folosi sunt:

- **char** reprezintă caractere ASCII; Se reţine în 8 biţi (1 octet)
- **int** reprezintă numere întregi în intervalul [-32767,32767]; Se reţine în 16 biţi
- **long int** reprezintă numere întregi în intervalul [-2147483647, 2147483647]; Se reţine în 32 biţi.
- **float** reprezintă numere în virgulă mobilă, cu o precizie de cel putin 7 zecimale exacte; Se reţine în 32 biţi.
- **double** reprezintă numere în virgulă mobilă, cu o precizie de cel puţin 15 zecimale exacte; Se reţine în 64 biţi.

O *variabilă* este numele unei locaţii din memorie utilizată pentru a păstra o valoare care poate fi modificată de program. Tipul fiecărei variabile trebuie declarat la începutul primului bloc de instrucţiuni în care ea este folosită. Variabile pot fi:

- *locale*, dacă sunt declarate în interiorul unei funcţii. Deşi de obicei toate variabilele unei funcţii sunt introduse la început, înaintea oricărei alte instrucţiuni, ele pot fi declarate şi la începutul unui bloc de instrucţiuni caz în care sunt accesibile doar instrucţiunilor blocului respectiv. Variabilele locale există atât timp cât este apelată funcţia sau blocul în care au fost declarate.

- *globale*, dacă sunt declarate în exteriorul oricărei funcţii. Aceste variabile sunt recunoscute în întreg programul dar este recomandabilă folosirea lor cât mai puţin pentru economie de memorie, evitare de confuzii şi o mai mare portabilitate.

O *matrice (vector)* este o colecţie de variabile de acelaşi tip care se declară sub forma

tip nume[*mărime 1*][*mărime 2*]...[*mărime p*];

Accesul la un element al matricii se face cu ajutorul indicilor. De exemplu, matricea

float mat[10][9];

are elementele mat[0][0], mat[0][1],..., mat[0][8] pe prima linie, mat[1][0], mat[1][1],..., mat[1][8] pe a doua linie s.a.m.d. Ultima linie a matricii, a zecea, este formată din elementele mat[9][0], mat[9][1],..., mat[9][8]. În C toate matricile au indici începând de la zero şi din această cauză trebuie avută multă grijă la declararea lor. Numărul de elemente ale unei matricii trebuie cunoscut la compilare.

Numele unei matrici, fără nici un indice, este un pointer care indică adresa primului element al matricii. În exemplul de mai sus, mat şi &mat[0][0] este acelaşi lucru, adresa primului element al matricii mat. De asemena, ∗(mat+9) şi mat[1][0] reprezintă acelaşi lucru, valoarea elementului al 10-lea, primul de pe linia a doua.

4 Principalele instrucţiuni ale limbajului

Algoritmul care stă la baza unui program este descris prin intermediul unor comenzi realizate de operatori sau instrucţiuni. Cei mai importanţi operatori sunt:

4.1 Operatorul de atribuire

$$\boxed{\textit{variabilă} = \textit{expresie};}$$

Variabilei din stânga i se atribuie valoarea specificată de expresia din partea dreaptă.

Exemplu:

x[1]:=M_PI+sqrt(3)+exp(4)+sin(1)+tan(2)+fabs(-1.2)+log(2)+pow(3.2,2.4)+pow10(9.8);

La atribuire, valoarea expresiei din membrul drept este convertită la tipul din membrul stâng.

4.2 Operatori aritmetici

Sunt operatorii care realizează operaţiile matematice simple:

- + adunare
- − scădere
- ∗ înmulţire

- / împărţire
- % modulo (restul unei împărţiri întregi)
- ++ incrementare
- −− decrementare

4.3 Operatori logici şi relaţionali

- && şi (and) logic
- || sau (or) logic
- ! negaţie (not) logică
- <, <=, >, >= mai mare sau mai mic (şi egal)
- == egal
- ! = diferit

4.4 Operatorii pentru pointeri

- & operator unar care returnează adresa din memorie a unei variabile.
- ∗ operator unar care se aplică unui pointer şi care returnează valoarea variabilei localizate la adresa respectivă.

Trecem acum la prezentarea celor mai importante instrucţiuni.

4.5 Instrucţiunea de ciclare prin numărare

for(*iniţializare* **;** *condiţie* **;** *incrementare***)**
{
*instrucţiuni***;**
}

Se execută repetat instrucţiunile cuprinse între acolade atât timp cât variabila de control a buclei îndeplineşte *condiţie* şi variază după cum indică *incrementare* începând de la valoarea din *iniţializare*.

Exemplu: Se calculează 10!

$x = 1;$
for$(j = 1; j <= 10; j + +)$
{
$x = x * j;$
}

În *incrementare* putem avea o lege $j + +$ dar şi $j - -$ sau $j+ = 2$. Variabila de control nu trebuie modificată în interiorul buclei. Putem avea mai multe variabile de control şi mai multe condiţii.

4.6 Instrucţiunea de repetare cu while

while (*condiţie*)
{
instrucţiuni;
}

Instrucţiunile situate între acolade se execută repetat atâta timp cât condiţia este adevărată. Când expresia logică din condiţie devine falsă se trece la executarea instrucţiunilor de după bucla **while**.

Exemplu: Se calculează suma numerelor de la 1 la 25.

$sum = 0;$
$j = 0;$
while$(j! = 26)$
}
$j++;$
$sum+ = j;$
};

Bucla **while** testează condiţia la început ceea ce înseamnă că nu se va executa corpul buclei dacă acea condiţie este iniţial falsă.

4.7 Instrucţiunea de repetare cu do-while

do
{
instrucţiuni
}
while (condiţie);

Instrucţiunile situate între acolade se execută repetat atâta timp cât condiţia este adevărată. Când expresia logică din condiţie devine falsă se trece la executarea instrucţiunilor de după bucla **do-while**.

Exemplu: Se calculează suma numerelor de la 1 la 25.

$sum = 0;$
$j = 0;$
do
{
$++j;$
$sum+ = j;$
}
while $(j <= 25);$

Bucla **do-while** testează condiţia la sfârşit, ceea ce înseamnă că se va executa cel puţin o dată corpul buclei, chiar dacă acea condiţie este iniţial falsă.

4.8 Instrucţiuni de salt

- **return** se foloseşte pentru revenirea dintr-o funcţie. În C++ o funcţie care nu este **void** trebuie neapărat să returneze o valoare prin expesia care urmează lui **return**.

- **break** se foloseşte într-o buclă pentru a o încheia imediat şi a se relua programul cu prima instrucţiune de după buclă.
- **continue** forţează trecerea la următoarea iteraţie a buclei, ignorând restul codului iteraţiei în care se află.
- **go to** are asociată o etichetă care indică o instrucţiune la care trebuie realizat saltul.
- **exit(0)** funcţie al cărui prototip este în stdlib.h şi care determină încheierea imediată a unui program.

4.9 Instrucţiunea de selecţie *if*

if (*condiţie*)
{
instrucţiuni 1;
}
else
{
instrucţiuni 2;
}

INSTRUCŢIUNILE 1 SITUATE ÎNAINTE DE **else** SE EXECUTĂ NUMAI DACĂ ESTE ADEVĂRATĂ CONDIŢIA. DACĂ ACEASTA ESTE FALSĂ SE TRECE LA EXECUTAREA INSTRUCŢIUNILOR 2 DE DUPĂ **else** DACĂ ACEASTA EXISTĂ SAU LA PRIMA INSTRUCŢIUNE DUPĂ **if** ÎN CAZ CONTRAR.

EXEMPLU: Se determină cel mai mare dintre numerele x şi y şi se reţine în max.

$max = 0$;
if $(x > y)$
{
$max = x$;
}
else
{
$max = y$;
}

Opţiunea **else** poate să nu apară. În acest caz instrucţiunile 1 situate după **if** se execută dacă se verifică condiţia. Se trece direct după **if** dacă expresia logică este falsă de la început. Testarea condiţiei se face înainte de prima executare a instrucţiunilor.

Este recomandabilă folosirea acoladelor pentru evitarea erorilor, mai ales la folosirea mai multor instrucţiuni **if**. De reţinut că **else** se referă la ultimul **if** în lipsa acoladelor. În cazul unei instrucţiuni de forma

if *(condiţie)* variabila=expresie 1
else variabila=expresie 2;

se poate scrie prescurtat

variabila = *(condiţie)* ? expresie 1 : expresie 2;

4.10 Pointeri

Un pointer este o variabilă care conţine o adresă din memorie. Dacă o variabilă urmează să reţină un pointer ea trebuie declarată astfel

tip * *nume_de_variabilă*;

unde *tip* este tipul variabilei indicată de pointer.

Există o strânsă legătură între matrici şi pointeri. Astfel dacă avem următoarea secvenţă

float a[10], *b;
b=a;

iniţial a este o matrice (un vector) de 10 elemente de tip float iar b un pointer la o variabilă float. Indentificatorul a reprezintă un pointer la primul element al matricii şi are loc relaţia $b = \&a[0]$.

4.11 Sistemul de intrare/ieşire (I/O)

Pentru introducerea de la tastatură şi afişarea pe ecran a rezultatelor în C++ se foloseşte fişierul antet **iostream.h** care este analogul lui **stdio.h** din C. El trebuie introdus prin directiva #**include** la începutul programului.

Introducerea datelor de la tastatură (echipamentul de intrare standard) se realiează prin cuvântul **cin** urmat de operatotul >>.

cin >> *identificator*$_1$ >> *identificator*$_2$ >> ... >> *identificator*$_p$**;**

Permite introducerea de la tastatură a datelor care se reţin pe rând în $identificator_1$, $identificator_2$,...$identificator_p$.

Pentru afişarea datelor pe ecran se foloseşte cuvântul **cout** şi operatorul <<.

cout << *identificator*$_1$ << *identificator*$_2$ << ... << *identificator*$_p$**;**

Permite scrierea pe ecranul utilizatorului a datelor reţinute în $identificator_1$, $identificator_2$,...,$identificator_p$.

Pentru afişarea cu un număr mare de zecimale se foloseşte intrucţiunea

cout.precision(r);

unde r indică numărul de zecimale dorit.

Pentru afişarea unui text se folosesc gilimelele, ″ ″. De asemenea, pentru trecerea pe un nou rând se utilizează ″|n″ sau **endl**.

5 Funcţii

Funcţiile sunt părţi de program închise în sine, constituite din date locale, instrucţiuni şi alte subfuncţii (structură asemănătoare cu cea a programului principal). Ele au câte un nume prin care pot fi apelate (activate). Funcţiile au avantajul unei mai bune structurări a textului sursă, care devine astfel mai clar şi capătă un caracter modular.

Sintaxa declaraţiei unei funcţii este următoarea:

tip nume_de_funcţie(*lista_parametrilor_formali*);
{
corpul funcţiei (bloc de instrucţiuni);
}

Nume_de_funcţie este un nume de program.

Lista_parametrilor_formali este o enumerare de nume de variabile cu tipurile corespunzătoare lor, separate prin virgulă, care primesc valorile argumentelor atunci când este apelată funcţia. Parametrii formali sunt variabile locale care pot fi referite în corpul funcţiei prin identificatorul lor. O funcţie poate să nu aibă parametri, caz în care lista lor este vidă. Chiar şi în acest caz, parantezele sunt necesare.

Tip se referă la tipul de date returnate de funcţie. O funcţie poate returna orice tip de date, cu excepţia unei matrice. Funcţia se va încheia cu intrucţiunea **return** urmată de valoarea corespunzătoare. Dacă nu se specifică nici un tip, el se consideră implicit **int**.

Există şi funcţii care nu returnează nici o valoare. Acestea au înainte cuvântul **void**. În acest caz **return** este folosit doar pentru a încheia funcţia şi a se reveni în programul apelant.

Corpul funcţiei este constituit din declaraţii de variabile locale şi instrucţiuni. El trebuie să funcţioneze ca un program de sine stătător.

Activarea funcţiei se face prin apelarea numelui ei urmat de lista parametrilor actuali. Corespondenţa între un parametru actual şi unul formal se face prin poziţia ocupată de acesta în cele două liste. Structurile celor două liste trebuie să fie identice iar tipul parametrilor actuali să coincidă cu tipul declarat al parametrilor formali corespunzători. Există două modalităţi de apelare a unei funcţii

- apelare prin valoare: la începutul activării funcţiei parametrii formali iau valorile parametrilor actuali. Modificările efectuate asupra parametrilor formali nu au efect asupra argumentului şi nu se reflectă în afara funcţiei.
- apelare prin referinţă: la începutul activării funcţiei parametrii formali iau adresele parametrilor actuali. Astfel, modificările asupra parametrului formal se reflectă în afara funcţiei.

Exemplu 1: Funcţia care calculează maximul a două numere.

```
float max(float x,float y);
{
float m;
if (x > y)
{
m = x;
}
else
{
m = y;
}
return m;
}
```

Exemplu 2: Funcţia care inversează două numere între ele.

```
void inv(float *x,float *y);
{
float m;
m = *x;
*x = *y;
*y = m;
}
```

Exemplu 3: Funcţia de calcul al produsului dintre o matrice şi un vector.

```
void prod(float a[10][10], float x[10], float y[10]);
{
float sum;
for(int i = 0; i <= 9; i + +)
{
sum=0;
for(int j = 0; j <= 9; j + +)
sum+=a[i][j]*x[j];
y[i]=sum;
}
}
```

```
/*Determina precizia cu care este retinun un numar in calculator
in simpla si duble precizie: machine zero*/
#include <iostream.h>
#include <conio.h>
float FloatMachineEps(); //Prototipuri de functii
double DoubleMachineEps();

void main() { float fmz; double dmz;
clrscr();
cout.precision(30);
fmz=FloatMachineEps();
dmz=DoubleMachineEps();
cout << "Machine's zero for single precision is: " <<fmz << endl;
cout << "Machine's zero for double precision is: " <<dmz<< endl; }

float FloatMachineEps(){ /*determina zeroul masinii in simpla precizie*/
float fmachine_z, ftest;
fmachine_z = 1.0;
ftest = 1.0 + fmachine_z;
while(1.0 != ftest){
fmachine_z = fmachine_z/2.0;
ftest = 1.0 + fmachine_z;
cout<<ftest<<"—n"; }
//cout<< "Eroarea maxima in precizie simpla este: "<< fmachine_z<<"—n";
return fmachine_z; }

double DoubleMachineEps(){ /*determina zeroul masinii in dubla precizie*/
double dmachine_z, dtest;
dmachine_z = 1.0;
dtest = 1.0 + dmachine_z;
while(1.0 != dtest){
dmachine_z = dmachine_z/2.0;
dtest = 1.0 + dmachine_z;
cout<< dtest<<"—n"; }
return dmachine_z; }
```