Programare in C++, POO

1. INTRODUCERE IN LIMBAJUL C++

1.1 Diferente intre limbajele C si C++

Limbajul C++ aduce o serie de inovatii fata de limbajul C standard care nu sunt legate direct de aparitia claselor: alt fel de comentarii, alt mod de afisare la consola, declaratii noi de parametri.

Cu ocazia adaugarii facilitatilor necesare POO s-au mai adus si alte imbunatatiri limbajului C, astfel incat C++ este un "C mai bun" si nu doar un "C cu clase" (de unde si numele limbajului).

Cu mici exceptii, exista compatibilitate intre cele doua limbaje, in sensul ca un program C este acceptat de compilatorul C++ si produce aceleasi rezultate la executie. Compilatoarele Borland (si altele) trateaza continutul unui fisier sursa in functie de extensia la numele fisierului: fisierele cu extensia CPP contin programe C++, iar fisiere cu orice alta extensie se considera a fi scrise in limbajul C.

O parte dintre inovatiile aduse sunt importante si pentru cei care nu folosesc clase in programele lor; de aceea ele vor fi prezentate pe scurt in cele ce urmeaza. Dintre aceste inovatii, cea mai importanta pentru programatori este folosirea de parametri referinta in loc de pointeri, pentru parametri modificabili de catre functii.

Comentarii

Pornind de la observatia ca multe comentarii se termina la sfarsit de linie, in C++ se accepta si comentarii care incep cu perechea de caractere "//" si se termina odata cu linia in care incep, fara alt terminator special de comentariu.

Declaratii

In C++ declaratiile de variabile sunt tratate la fel cu instructiunile si deci pot apare oriunde intr-un bloc; in C declaratiile trebuie sa preceada prima instructiune executabila dintr-un bloc. Se poate vorbi chiar un alt stil de programare, in care o variabila este declarata acolo unde este folosita prima data. Exemplu:

```
// suma valorilor dintr-un vector
float sum (float x[], int n )
{
	float s=0;
	for (int i=0; i<n; i++)
		s += x[i];
	return s;
}
```

Domeniul de valabilitate al variabilei este blocul unde a fost declarata variabila, dar instructiunea "for" prezinta un caz special.

In versiunile mai noi ale limbajului C++ domeniul de valabilitate al variabilei declarate intr-o instructiune "for" este limitat la instructiunile care vor fi repetate (din cadrul ciclului "for"). Din acest motiv secventa urmatoare poate produce sau nu erori sintactice:

```
...
  for (int i=0;i<6;i++) a[i]=i;
  for (int i=0;i<6;i++) printf ("%d ", a[i];
```

Structuri

In C++ se admite folosirea numelui unui tip structura, fara a fi precedat de "struct" si fara a mai fi necesar "typedef ". Exemplu:

```
struct nod{
       int val;
       nod * leg;     // in C: struct nod * leg
    };
nod * lista;      // in C: struct nod * lista
```

Functii

In C++ toate functiile folosite trebuie declarate si nu se mai considera ca o functie nedeclarata este implicit de tipul "int". Dar o functie definita fara un tip explicit este considerata ca fiind de tip "int". Asa se explica de ce functia "main" este deseori declarata ca fiind de tip " void "; absenta cuvintului "void " implica tipul " int" pentru functia "main" si se verifica existenta unei instructiuni "return" cu expresie de tip intreg.

In C++ se pot declara valori implicite pentru parametrii formali de la sfarsitul listei de parametri; aceste valori sunt folosite automat in absenta parametrilor efectivi corespunzatori la un apel de functie. O astfel de functie poate fi apelata deci cu un numar variabil de parametri. Exemplu:

```
// afisare vector, precedata de un titlu
 void printv ( int v[], int n, char * titlu="") {
       printf ("\n %s \n", titlu);  // daca sir nenul
    for (int i=0; i<n; i++)
       printf ("%d ", v[i]);
 }
    ...
   // exemple de apeluri
 printv ( x,nx );              // cu 2 parametri
 printv (a,na," multimea A este");   // cu 3 parametri
```

In C++ functiile scurte pot fi declarate "inline", iar compilatorul inlocuieste apelul unei functii "inline" cu instructiunile din definitia functiei, eliminand secventele de transmitere a parametrilor. Functiile "inline" sunt tratate ca si macrourile definite cu "define". Orice functie poate fi declarata "inline", dar compilatorul poate decide ca anumite functii nu pot fi tratate "inline" si sunt tratate ca functii obsnuite. De exemplu, functiile care contin cicluri nu pot fi "inline".

Exemplu de functie "inline":

```
inline int max (int a, int b) {
    return a>b ? a : b;
}
```

Utilizarea unei functii "inline" nu se deosebeste de aceea a unei functii normale.

In C++ pot exista mai multe functii cu acelasi nume dar cu parametri diferiti (ca tip sau ca numar). Se spune ca un nume este "supraincarcat" cu mai multe semnificatii ("function overloading"). Compilatorul poate stabili care din functiile cu acelasi nume a fost apelata intr-un loc analizind lista de parametri si tipul functiei. Exemplu:

```
int cub (int n) { return n*n*n; }
float cub (float f) { return f*f*f; }
double cub (double d) { return d*d*d; }
...
printf ("%d %f %f \n", cub(2), cub(2.5f), cub (2.5) );
```

Supradefinirea se practica pentru functiile membre (din clase) si, in particular, pentru operatori definiti fie prin functii membre, fie prin functii prietene.

1.2 **Operatori noi**

In C++ s-au introdus doi operatori noi, pentru alocare dinamica a memoriei ("new ") si pentru eliberarea memoriei dinamice ("delete "), destinati sa inlocuiasca functiile de alocare si eliberare (malloc, free, s.a.). Operatorul "new " are ca operand un nume de tip, urmat in general de o valoare initiala pentru variabila creata (intre paranteze rotunde); rezultatul lui "new" este o adresa (un pointer de tipul specificat) sau NULL daca nu exista suficienta memorie libera.
Exemple:

```
nod * pnod;         // pointer la nod de lista
pnod = new nod;     // alocare fara initializare
assert (pnod != NULL);
...
int * p = new int(3);  // alocare cu initializare
```

Operatorul "new" are o forma putin modificata la alocarea de memorie pentru vectori, pentru a specifica numarul de componente:

```
int * v = new int [n];
...
for (int i=0; i<n; i++) v[i]=0;
```

De remarcat ca nu se pot initializa componentele unui vector alocat cu "new" chiar in expresia care urmeaza lui "new".

Operatorul " delete" are ca operand o variabila pointer si are ca efect eliberarea blocului de memorie adresat de pointer, a carui marime rezulta din tipul variabilei pointer sau este indicata explicit. Exemple:

```
int * v;
delete v;       // elibereaza sizeof(int) octeti
delete [] v;    // dimensiunea blocului este memorata in antetul lui
delete [n] v;   // elibereaza n*sizeof(int) octeti
```

Operatorul de rezolutie "::" este necesar pentru a preciza domeniul de nume caruia ii apartine un nume de variabila sau de functie.

Fiecare clasa creaza un domeniu separat pentru numele definite in acea clasa (pentru membri clasei). Deci un acelasi nume poate fi folosit pentru o variabila externa (definita in afara claselor), pentru o variabila locala unei functii sau pentru o variabila membra a unei clase. Exemplu:

```
int end;            // variabila externa
void cit () {
  int end=0;        // variabila locala
...
  if (::end) { ...}  // variabila externa
}
class A {
 public:
   int end;         // variabila membru a clasei A
   void print();
 ...
};
// exemple de utilizare in "main"
 end=1;     // sau
```

```
A::end=0;
f.seekg (0, ios::end);   // din clasa predefinita "ios"
...
```

Utilizarea operatorului de rezolutie este necesara la definirea metodelor unei clase in afara clasei, pentru a preciza compilatorului ca este definitia unei metode si nu definitia unei functii externe.

Exemplu:

```
  // definitie metoda din clasa A
void A::print () { ... }
  // definitie functie externa
void print () { ... }
```

1.3 **Tipuri referinta**

In C++ s-au introdus tipuri referinta, folosite in primul rand pentru parametri modificabili sau de dimensiuni mari. Si functiile care au ca rezultat un obiect mare pot fi declarate de un tip referinta, pentru a obtine un cod mai performant. Caracterul ampersand (&) folosit dupa tipul si inaintea numelui unui parametru formal (sau unei functii) arata compilatorului ca pentru acel parametru se primeste adresa si nu valoarea argumentului efectiv. Spre deosebire de un parametru pointer, un parametru referinta este folosit de utilizator in interiorul functiei la fel ca un parametru transmis prin valoare, dar compilatorul va genera automat indirectarea prin pointerul transmis (in programul sursa nu se foloseste explicit operatorul de indirectare '*'). Exemplu:

```
    // schimba intre ele doua valori
void schimb (int & x, int & y) {
   int t = x;
   x = y; y = t;
}
// ordonare vector
void sort ( int a[], int n ) {
  ...
  if ( a[i] > a[i+1])
    schimb ( a[i], a[i+1]);
  ...
}
```

Referintele simplifica utilizarea unor parametri modificabili de tip pointer, eliminand necesitatea unui pointer la pointer. Exemplu:

```
void initL ( nod* & cap) {
  cap=new nod;    // cap este un pointer
  cap->leg=NULL:
}
```

Sintaxa declararii unui tip referinta este urmatoarea:

tip & nume

unde "nume" poate fi:
- numele unui parametru formal
- numele unei functii (urmat de lista argumentelor formale)
- numele unei variabile (mai rar)

Efectul caracterului '&' intr-o declaratie este urmatorul: compilatorul creaza o variabila pointer la tipul lui "nume", o initializeaza cu adresa asociata lui "nume" si retine ca orice referire ulterioara la "nume" va fi tradusa printr-o indirectare prin variabila pointer anonima creata.

O functie poate avea ca rezultat o referinta la un vector dar nu poate avea ca rezultat un vector.

O functie nu poate avea ca rezultat o referinta la o variabila locala.

Exemple:

```
typedef int vec [M];
  // adunarea a 2 vectori - gresit !
vec& suma (vec a, vec b, int n) {
  vec c;
  for (int i=0; i<n;i++)
     c[i]=a[i]+b[i];
  return c;  // eroare !!!
}
```

1.4 **Fluxuri de intrare-iesire**

In C++ s-a introdus si o alta posibilitate de exprimare a operatiilor de citire-scriere, pe langa folosirea functiilor standard de intrare-iesire din limbajul C. In acest scop se folosesc citeva clase predefinite pentru "fluxuri de I/E" (declarate in fisierele antet `iostream.h` si `fstream.h`).

Un flux de date ("stream") este un obiect care contine datele si metodele necesare operatiilor cu acel flux.

Pentru operatii de I/E la consola sunt definite trei variabile de tip flux, numite "cin" (console input), "cout" (console output) si "cerr" (console errors). Pentru aceste fluxuri nu mai sunt necesare operatii de deschidere si de inchidere.

Operatiile de citire sau scriere cu un flux pot fi exprimate prin metode ale claselor flux sau prin doi operatori cu rol de extracator din flux (>>) sau insertor in flux (<<). Atunci cind primul operand este de un tip flux, interpretarea acestor operatori nu mai este cea de deplasare binara ci este extragerea de date din flux (>>) sau introducerea de date in flux (<<).

Operatorii << si >> implica o conversie automata a datelor intre forma interna (binara) si forma externa (sir de caractere). Formatul de conversie poate fi controlat prin cuvinte cheie cu rol de "modificator".

Exemplu:

```
#include <iostream.h>
void main () {
   int n; float f; char s[20];
   cout << " n= ";  cin >> n;
   cout << " f= ";  cin >> f;
   cout << " un sir: "; cin >> s;  cout << s << "\n";
}
```

Intr-o expresie ce contine operatorul << primul operand trebuie sa fie "cout" (sau o alta variabila de un tip "`ostream` "), iar al doilea operand poate sa fie de orice tip aritmetic sau de tip "`char*`" pentru afisarea sirurilor de caractere. Rezultatul expresiei fiind de tipul primului operand, este posibila o expresie cu mai multi operanzi (ca la atribuirea multipla).

Exemplu:

```
cout << "x= " << x << "\n";
```

este o prescurtare a secventei de operatii:

```
cout << "x= "; cout << x; cout << "\n";
```

In mod similar, intr-o expresie ce contine operatori >> primul operand trebuie sa fie "cin" sau de un alt tip "istream ", iar ceilalti operanzi pot fi de orice tip aritmetic sau pointer la caractere.

Exemplu:

```
cin >> x >> y;
```

este echivalent cu secventa:

```
cin >> x; cin >> y;
```

Operatorii << si >> pot fi incarcati si cu alte interpretari, pentru scrierea sau citirea unor variabile de orice tip clasa, cu conditia (supra)definirii lor in clasele respective.

Operatii cu fisiere in C++

Un fisier este vazut in C++ ca un obiect, deci ca o variabila de un tip clasa. Pentru fisiere oarecare se pot folosi 3 clase predefinite:

- fstream pentru fisiere ce pot fi folosite sau in citire sau in scriere
- ifstream pentru fisiere din care este permisa doar citirea
- ofstream pentru fisiere in care este permisa doar scrierea

Constructorul claselor ifstream si ofstream trebuie sa primeasca ca parametru numele fisierului (un sir de caractere, care include si calea), dar mai poate avea si un al doilea parametru intreg care specifica anumite optiuni de lucru cu fisierul.

Efectul operatorilor de insertie si de extractie este similar cu cel al functiilor standard "fprintf" si "fscanf", dar nu trebuie precizat formatul de conversie (care rezulta din tipul variabilelor folosite), iar la citire nu trebuie folositi pointeri (ci direct variabilele pentru care se citesc valori). Exemplu:

```
// crearea unui fisier de numere intregi
#include <fstream.h>
void main () {
  ofstream out ("test.dat");
  for (int k=1;k<11;k++)
    out << k << " ";
  out.close();  // poate lipsi in acest caz
}
```

Fisierul creat de programul anterior poate fi afisat cu orice editor de texte sau folosind programul urmator:

```
// citire fisier de numere intregi
#include <fstream.h>
void main () {
 int x;
 ifstream in ("teste.dat");
 while ( ! in.eof() ) {
   in >> x; cout << x << "\n";
 }
}
```

Metoda "eof()" face parte dintr-un grup de metode care permit citirea starii unui flux si verificarea anumitor conditii, inclusiv a modului cum s-a terminat ultima operatie cu fluxul respectiv:

```
int eof ();   // rez. nenul daca sfirsit de fisier
```

```
int good();   // rez. nenul daca operatie terminata bine
int bad();    // rez. nenul daca s-a produs o eroare
int fail();   // rez. nenul daca operatia nu a reusit
void clear (int=0);  // sterge sau modifica cuvintul de stare
int rdstate();  // citeste cuvintul de stare
```

Primele 4 functii citesc cuvintul de stare si testeaza anumiti biti din acest cuvint, ceea ce se poate face si explicit in program.

Programul urmator face citeva verificari suplimentare fata de exemplul anterior.

```
// citire din fisier, cu verificari
#include <fstream.h>
void main () {
 int x;
 ifstream in ("teste.dat");
 if ( in.fail()) {
   cout << "eroare la deschidere !\n"; return ;
 }
 while ( ! in.eof() ) {
   in >> x;
   if ( in.good())
     cout << x << "\n";
 }
}
```

Alte metode ale claselor flux permit operatii de citire-scriere ce nu pot fi realizate prin operatorii de insertie si extractie:

```
getline ( char * buf, int bufsize, char ='\n')
```

Citirea unei linii de lungime maxima "bufsize", la adresa "buf", folosind ca terminator de linie caracterul '\n ' (implicit) sau alt caracter dat ca ultim parametru.

```
read ( unsigned char * buf, int bufsize);
write ( unsigned char * buf, int bufsize);
```

Citirea / scrierea unui bloc de "bufsize" octeti la(de la) adresa "buf".

Exemplul urmator arata cum se poate crea si folosi un fisier binar de numere in format intern.

```
#include <fstream.h>
void main () {
  float x;
  ofstream out ("test",ios::binary||ios::out);
  if ( out.fail() ) {
    cout << " eroare la creare fisier \n"; return;
  }
  for (x=1;x<10;x=x+0.2)
    out. write ((unsigned char *)&x, sizeof (x));
  out.close();
  ifstream in ("test",ios::binary);
  if ( in.fail() ) {
    cout << " eroare la deschidere fisier \n"; return;
  }
  cout << "\n citite din fisier: \n";
  while (! in.eof()) {
    in. read ((unsigned char *)&x, sizeof (x));
    if (in.good())
```

```
        cout << x << " ";
    }
}
```

Alte metode ale claselor flux permit accesul direct prin pozitionare in flux si prin determinarea pozitiei curente, la citire sau la scriere.

Metoda "rdbuf()" are ca rezultat un pointer la zona tampon ("buffer") a unui flux, pointer care se poate folosi pentru citirea continutului zonei buffer (deci a fisierului).