LUCRARE DE LABORATOR NR. 1

Tema: Clase și obiecte. Constructorii și destructorul clasei

Scopul lucrării:

- Studierea principiilor de definire şi utilizare a constructorilor
- Studierea principiilor de definire şi utilizare a destructorilor
- Studierea tipurilor de constructori

Noţiuni de bază

Una din cele mai răspîndite erori de programare (în orice limbaj) constă în utilizarea obiectelor fără iniţializare anterioară, aşa precum nu toate limbajele asigură iniţializare automată. Desigur, poate fi definită funcţia de iniţializare şi de distrugere a obiectului:

```
class Book{
char *author;
int year;
int pages;
public:
   void Init(char*, int, int);
   void Destroy();
};
void Book::Init(char* a, int y, int p){
   author = new char[strlen(a)+1];
   strcpy(author,a);
   year=y;
pages=p;
}
void Book::Destroy(){
   delete[] author;
}
```

În acest exemplu, însă, nimeni nu garantează că iniţializarea va avea loc, sau că va fi eliberată memoria alocată. Alt neajuns al exemplului dat constă în pericolul scurgerii de memorie, deoarece funcţia de iniţializare poate fi apelată de nenumărate ori. De asemenea se poate bloca sistemul din cauza utilizării eronate a memoriei dinamice, motivul fiind apelul funcţiei `Destroy` fără iniţializare.

Pentru a evita această eroare, C++ asigură mecanismul de iniţializare automată pentru clasele definite de utilizator – *constructorul* clasei. Iar pentru operaţiile de distrugere – *destructorul* clasei.

Constructorul – este o funcţie membru specială, de acelaşi nume cu numele clasei, care se apelează automat la crearea obiectului de tipul clasei. Constructorul nu returnează nici un rezultat, chiar nici *void*. Compilatorul garantează apelarea unică a constructorului pentru un singur obiect.

Destructorul – este o funcţie membru specială, care se apelează automat la distrugerea obiectului. Numele unui destructor este numele clasei precedat de caracterul „~”.Compilatorul garantează apelarea unică a destructorului pentru un singur obiect. Un destructor nu are parametri, de aceea nu poate fi supraîncărcat, şi el este unic pentru o clasă.

Pentru un obiect local, destructorul se apelează cînd controlul programului iese din domeniul lui (se iese din blocul în care este declarat). Obiectele dinamice nu pot fi distruse automat. Distrugerea se realizează de către programator prin intermediul operatorului *delete,* care apelează, propriu zis, destructorul.

Fie un exemplu analogic cu cel precedent, utilizînd constructorul şi destructorul:

```
class Book{
char *author;
int year;
int pages;
public:
      Book(char*, int, int);
      ~Book();
};
Book::Book(char* a, int y, int p){
      author = new char[strlen(a)+1];
      strcpy(author,a);
      year=y;
pages=p;
}
Book::~Book(){
      delete[] author;
}
void main(){
    Book b("Stroustrup",2000,890);
     // Book b1;  // încercarea de a crea obiectul fără apelul
                  //constructorului duce la erori

     Book* ptr = new Book("Lippman",1996, 1200);

                  // crearea dinamică a obiectului
      delete ptr;
                  // în acest caz eliberarea memoriei se realizează de
                  //către programator
      }
// aici se apelează automat destructorul pentru b.
```

Tipurile constructorilor

Există patru tipuri de constructori: implicit, de copiere, de conversie a tipului şi general. Această clasificare este bazată pe regulile de definire şi utilizare a constructorilor.

- Constructorul implicit – constructor fără parametri, sau constructor cu toţi parametrii impliciţi.
- Constructorul de copiere – constructor care are ca parametru referinţă la obiectul clasei respective. Constructorul de copiere poate avea şi alţi parametri care însă trebuie să fie impliciţi.
- Constructorul de conversie a tipului - constructorul, care transformă un tip de obiect în altul.
- Constructorul general – constructorul care nu corespunde categoriilor enumerate mai sus.

Fie exemplul:

```
class Book{
char *author;
int year;
int pages;
public:
      Book();                 // constructor implicit
      Book(const Book&);      // constructor de copiere
      Book(const char*);      // constructor de conversie a tipului
      Book(char*, int, int);  // constructor general
      ...
};
...
void main(){
 Book b("Stroustrup",1997,890);     // general
 Book b1 = b, b11(b);               // de copiere
 Book b2 = "Stroustrup", b21("Bjarne");
 // de schimbare a tipului
 Book b3;                           // implicit
}
```

Constructori generaţi de compilator. Interacţiunea lor cu constructorii definiţi de utilizator

Prezenţa constructorilor nu este obligatorie. Se pot defini clase şi fără constructori. În acest caz, compilatorul C++ generează în mod automat un constructor implicit şi de copiere.

Constructorul implicit generat nu realizează absolut nimic, iar constructorul de copiere generat automat implică copierea bit cu bit, ceea ce nu întotdeauna este satisfăcător. În cazul definirii unui constructor de către utilizator se generează automat numai constructorul de copiere.

Recomandări

Fiecare clasă trebuie să conţină constructori. Dacă clasa conţine elemente pointeri trebuie neapărat să se supraîncarce constructorul de copiere, care se utilizează pentru crearea copiilor obiectului, la apelul obiectului în funcţie. Motivul supraîncărcării constructorului de copiere constă în necesitatea utilizării acţiunilor suplimentare în comparaţie cu algoritmul standard de copiere, deoarece se poate întîmpla ca două obiecte diferite să refere una şi aceeaşi memorie, ceea ce poate duce la pierderea informaţiei şi la erori în sistemul de operare.

Exemplu de constructori de copiere:

```
Book::Book(const Book& b){
cout<<"constructor de copiere"<<endl;
author = new char[strlen(b.author)+1];
strcpy(author, b.author);
 year = b.year;
 pages = b.pages;
}
```

Constructorii trebuie să utilizeze mecanisme numite iniţializatori (lista de iniţializare). Această regulă contribuie la evitarea cheltuielilor necesare pentru iniţializarea cîmpurilor clasei, deoarece la execuţia constructorului cîmpurile trebuie să fie deja iniţializate, iar la execuţia iniţializatorilor – nu.

```
Book::Book(const Book& b): year(b.year),pages(b.pages),
         author(new char[strlen(b.author)+1])
{
         strcpy(author, b.author);
}
   // apelul funcţiei strcpy a rămas în afara iniţializatorului.
```

În cazul utilizării tabloului de obiecte se defineşte în mod obligatoriu un constructor implicit, care se apelează pentru iniţializarea elementelor tabloului.

Întrebări de control:

1. Explicaţi termenul de iniţializare.
2. Ce cuvinte cheie se utilizează pentru definirea constructorului şi a destructorului?
3. Poate oare un constructor să returneze o valoare?
4. Pot oare să existe clase fără constructori?
5. Cîţi constructori poate conţine o clasă? Dar destructori? De ce?
6. Enumeraţi tipurile de constructori?
7. Cum se apelează un constructor?
8. De ce este nevoie de definit un constructor de copiere?
9. De ce este nevoie de definit un constructor de conversie a tipului?
10. În ce cazuri se apelează constructorul de copiere? Care este sintaxa apelului?
11. Ce sunt listele de iniţializare (iniţializatori)?
12. În care caz se defineşte obligatoriu constructorul implicit?

Sarcina

Varianta 1

Să se creeze clasa *Document* – document, care conţine informaţia despre denumirea, tema, autorul documentului utilizînd memoria dinamică; numărul de pagini, data şi timpul ultimei redactări. Să se definească toţi constructorii. Constructorul de definește conversie a tipului are ca parametru denumirea documentului. Să se definească funcţiile de modificare a temei, datei ultimei redactări ş. a.

Varianta 2

Să se creeze clasa *Image* – imagine, care conţine următoarea informaţie: denumirea fişierului, formatul de compresie, dimensiunile imaginii, dimensiunea în octeţi, compresia (în %). Să se definească toţi constructorii. Constructorul de conversie a tipului are ca parametru denumirea fişierului. Să se definească funcţiile de modificare a denumirii de fişier, a formatului, a dimensiunii ş. a.

Varianta 3

Să se creeze clasa *Queue* – coadă de tip *float*. Cîmpurile – numărul de elemente şi un pointer pentru alocarea dinamică a memoriei. Să se definească constructorii: implicit, de copiere şi cu un parametru – numărul necesar de elemente; funcţiile *add* şi *get* pentru punerea unui element în coadă şi pentru scoaterea unui element din coadă respectiv; funcţiile: *isEmpty,* care returnează valoarea 1 dacă coada este vidă şi zero în caz contrar, şi *isFull* care returnează valoarea 1 dacă coada este plină şi zero în caz contrar.

Varianta 4

Să se creeze clasa *String* – şir, utilizînd memoria dinamică. Să se definească constructorii: implicit, de copiere şi cu un parametru – pointer spre un şir de tip *char*. Să se definească funcţiile de atribuire a unui şir la altul, de comparaţie, de căutare a unui subşir, de numărare a simbolurilor ş. a.

Varianta 5

Să se creeze clasa *Discipline* - disciplină, care conţine informaţia despre denumirea obiectului, numele profesorului, numărul de ore la disciplină şi forma de susţinere (examen sau colocviu). Să se utilizeze memoria dinamică pentru setarea cîmpurilor textuale. Să se definească constructorii: implicit, de copiere şi cu un parametru – denumirea disciplinei. Să se definească funcţiile de schimbare: a profesorului, a numărului de ore şi a formei de susţinere.

Varianta 6

Să se creeze clasa *Soft* – fişier, care conţine informaţia despre numele complet al fişierului şi anexa de asociere (doc – Word, psd – Photoshop, etc.), utilizînd memoria dinamică, mărimea, data şi timpul creării. Să se definească toţi constructorii. Constructorul de conversie a tipului are parametrul – numele de fişier. Să se definească funcţiile de redenumire a fişierului, şi de modificare a anexei de asociere.

Varianta 7

Să se creeze clasa *Group* – grupa, care conţine informaţia despre codul grupei, specialitatea, numărul de studenţi, curatorul grupei şi anul de studii. Să se utilizeze memoria dinamică pentru setarea cîmpurilor textuale. Să se definească: toate tipurile de constructori; funcţiile de schimbare a curatorului şi a numărului de studenţi. Să se definească funcţia de setare a anului de studii în aşa fel, încît valoarea acestui cîmp să poată fi mărită şi să nu depăşească cifra 5.

Varianta 8

Să se creeze clasa *Set* – mulţimea numerelor întregi, utilizînd memoria dinamică. Să se definească constructorii: implicit şi de copiere. Să se definească funcţiile: de adăugare a unui element nou în mulţime; de determinare a apartenenţei elementului la mulţime; de adunare, scădere şi intersecţie a două mulţimi.

Varianta 9

Să se creeze clasa *Date* – dată cu cîmpurile: zi(1-28..31), lună(1-12), an (numere întregi). Să se definească constructorii; funcţiile membru de setare a zilei, lunii şi anului; funcţiile membru de returnare a zilei, lunii, anului; funcţiile de afişare: afişare tip „ 6 iunie 2004” şi afişare tip „6.06.2004”. Funcţiile de setare a cîmpurilor clasei trebuie să verifice corectitudinea parametrilor primiţi.

Varianta 10

Să se creeze clasa *Time* – timp cu cîmpurile: ore (0-23), minute (0-59), secunde (0-59). Să se definească constructorii, funcţiile membru de setare a timpului; de returnare a orei, minutelor şi secundelor; funcţiile de afişare conform şablonului: „ora 17 şi 18 minute şi 4 secunde” şi „5 p.m. 18 minute 4 secunde”. Funcţiile de setare a cîmpurilor clasei trebuie să verifice corectitudinea parametrilor primiţi.

Varianta 11

Să se creeze clasa *Stack* – stivă de tip *int*. Cîmpurile – numărul general şi numărul utilizat de elemente şi pointer pentru alocarea dinamică a memoriei. Să se definească constructorii: implicit, de copiere şi cu parametru – numărul necesar de elemente; funcţiile *Push* şi *Pop* pentru punerea unui element pe stivă şi scoaterea unui element din stivă respectiv; funcţiile *IsEmpty* şi *IsFull* pentru determinarea stării stivei. Funcţia *isEmpty* returnează valoarea 1 dacă

stiva este vidă şi zero în caz contrar, iar funcţia *isFull* returnează valoarea 1 dacă stiva este plină şi zero în caz contrar.

Varianta 12

Să se creeze clasa *Set* – mulţimea valorilor de tip *float*, utilizînd memoria dinamică. Să se definească constructorii: implicit şi de copiere. Să se definească funcţiile: de adăugare a unui element nou în mulţime; de determinare a apartenenţei elementului la mulţime; de adunare, scădere şi intersecţie a două mulţimi.

Varianta 13

Să se creeze clasa *Matrix* – matrice. Clasa conţine pointer spre *int*, numărul de rînduri şi de coloane şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru – matrice pătrată şi constructorul cu doi parametri – matrice dreptunghiulară. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a două matrice; înmulţirea unei matrice cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor matricelor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

Varianta 14

Să se creeze clasa *Vector* – vector. Clasa conţine pointer spre *double*, numărul de elemente şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru şi constructorul cu doi parametri. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a doi vectori; înmulţirea scalară unui vector cu un alt vector; înmulțirea unui vector cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor vectorilor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

Varianta 15

Să se creeze clasa *Matrix* – matrice. Clasa conţine pointer spre *float*, numărul de rînduri şi de coloane şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru – matrice pătrată şi constructorul cu doi parametri – matrice dreptunghiulară. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a două matrice; înmulţirea unei matrice cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor matricelor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

Varianta 16

Să se creeze clasa *Vector* – vector. Clasa conţine pointer spre *long,* numărul de elemente şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru şi constructorul cu doi parametri. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a doi vectori; înmulţirea scalară unui vector cu un alt vector; înmulţirea unui vector cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor vectorilor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

Varianta 17

Să se creeze clasa *Stack* – stivă de tip *double*. Cîmpurile – numărul general şi numărul utilizat de elemente şi pointer pentru alocarea dinamică a memoriei. Să se definească constructorii: implicit, de copiere şi cu parametru – numărul necesar de elemente; funcţiile *Push* şi *Pop* pentru punerea unui element pe stivă şi scoaterea unui element din stivă respectiv; funcţiile *IsEmpty* şi *IsFull* pentru determinarea stării stivei. Funcţia *isEmpty* returnează valoarea 1 dacă stiva este vidă şi zero în caz contrar, iar funcţia *isFull* returnează valoarea 1 dacă stiva este plină şi zero în caz contrar.

Varianta 18

Să se creeze clasa *Matrix* – matrice. Clasa conţine pointer spre *double*, numărul de rînduri şi de coloane şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru – matrice pătrată şi constructorul cu doi parametri – matrice dreptunghiulară. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a două matrice; înmulţirea unei matrice cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor matricelor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

Varianta 19

Să se creeze clasa *Vector* – vector. Clasa conţine pointer spre *int, i*numărul de elemente şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru şi constructorul cu doi parametri. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a doi vectori; înmulţirea scalară unui vector cu un alt vector; înmulţirea unui vector cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor vectorilor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

Varianta 20

Să se creeze clasa *Set* – mulţimea valorilor de tip *long*, utilizînd memoria dinamică. Să se definească constructorii: implicit şi de copiere. Să se definească funcţiile: de adăugare a unui element nou în mulţime; de determinare a apartenenţei elementului la mulţime; de adunare, scădere şi intersecţie a două mulţimi.

Varianta 21

Să se creeze clasa *Vector* – vector. Clasa conţine pointer spre *float,* numărul de elemente şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru şi constructorul cu doi parametri. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a doi vectori; înmulţirea scalară unui vector cu un alt vector; înmulţirea unui vector cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor vectorilor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

Varianta 22

Să se creeze clasa *Matrix* – matrice. Clasa conţine pointer spre *long*, numărul de rînduri şi de coloane şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru – matrice pătrată şi constructorul cu doi parametri – matrice dreptunghiulară. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a două matrice; înmulţirea unei matrice cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor matricelor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.