Ministerul Educației, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

###### Raport

Lucrare de laborator nr.3

Programare in C++

VARIANTA 19

A efectuat:

st. gr. SI-211 Perevoznic Vladislav

A verificat:

asist., univ. Scrob Sergiu

Chisinau 2022

# lucrare de laborator nr. 3

**Tema:** Supraîncărcarea operatorilor

**Scopul lucrării:**

* Studierea necesităţii supraîncărcării operatorilor;
* Studierea sintaxei de definire a operatorilor;
* Studierea tipurilor de operatori;
* Studierea formelor de supraîncărcare;

**Noţiuni de bază**

Avantajul utilizării operatorilor (reducerea codului) în acelaşi timp complică înţelegerea codului, deoarece nu este posibil întotdeauna sau este complicat să se urmărească dacă se utilizează un operator predefinit sau supraîncărcat. Însă supraîncărcarea operatorilor este necesară. Sunt probleme care se soluţionează numai prin supraîncărcarea operatorilor,cum ar fi operatorul de atribuire. Desigur, compilatorul poate să genereze codul necesar, sau să definim metoda, de exemplu Assign(Java). Dar ambele soluţii nu sunt ideale. Neajunsul primei metode constă în utilizarea copierii bit cu bit, ceea ce este satisfăcător atît timp cît în clasă nu se utilizează pointerii:

**Varianta 9**

а) Să se creeze clasa Time – timpul, care conţine cîmpurile: ore, minute, secunde. Să se definească operatorii "+" şi "-", ca funcţii prietene, iar "++" şi "--" în ambele forme (prefixă şi postfixă) ca metode ale clasei. Operatorii trebuie să permită realizarea operaţiilor atît cu variabilele clasei date, cît şi cu variabilele de tip predefinit int (desemnează secundele). b) Să se creeze clasa Queue – coadă, utilizînd memoria dinamică. Să se definească operatorii "+" – de adunare a două cozi, "=" – de atribuire ca metode ale clasei. Să se definească operatorii de comparare "==", "!=", "<", ">" ca funcţii prietene. Pentru realizarea ultimilor doi operatori să se definească funcţia de calcul a normei elementelor cozii. Să se definească operatorii "<<" şi ">>" pentru ieşiri/intrări de obiecte, precum şi pentru inserarea/eliminarea elementelor în/din coadă.

Codul programului (a) :

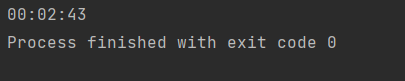
#include <iostream>  
#include <string>  
using namespace std;  
  
class Time{  
private:  
 int h,m,s;  
public:  
 Time(long int s){  
 this->init(s);  
 }  
 init(long int s)  
 {  
 if (s<0)s\*=-1;  
 this->h=s/3600;  
 this->m=(s%3600)/60;  
 this->s=s%60;  
 }  
 long int seconds(){  
 return h\*3600+m\*60+s;  
 }  
 string str(){  
 string hh="",mm="",ss="";  
 if(h<10)hh='0';  
 if(m<10)mm='0';  
 if(s<10)ss='0';  
 return hh+to\_string(h)+":"+  
 mm+to\_string(m)+":"+  
 ss+to\_string(s);  
 }  
 Time& operator++(){  
 Time aux = Time(this->seconds());  
 this->init(aux.seconds()+1);  
 return aux;  
 }  
 Time operator++(int){  
 Time aux = Time(this->seconds()+1);  
 this->init(aux.seconds());  
 return aux;  
 }  
 Time& operator--(){  
 Time aux = Time(this->seconds());  
 this->init(aux.seconds()-1);  
 return aux;  
 }  
 Time operator--(int){  
 Time aux = Time(this->seconds()-1);  
 this->init(aux.seconds());  
 return aux;  
 }  
 friend Time operator+(Time,Time);  
 friend Time operator+(Time a,int s);  
 friend Time operator-(Time a,Time b);  
 friend Time operator-(Time a,int s);  
};  
  
Time operator+(Time a,Time b){  
 return Time(a.seconds()+b.seconds());  
}  
  
Time operator+(Time a,int s){  
 return Time(a.seconds()+s);  
}  
  
Time operator-(Time a,Time b){  
 return Time(a.seconds()-b.seconds());  
}  
  
Time operator-(Time a,int s){  
 return Time(a.seconds()-s);  
}  
  
int main()  
{  
 Time a = Time(61);  
 Time b = Time(100);  
 a++;  
 ++b;  
 cout<<(a+b).str();  
 return 0;  
}

Codul programului (b) :

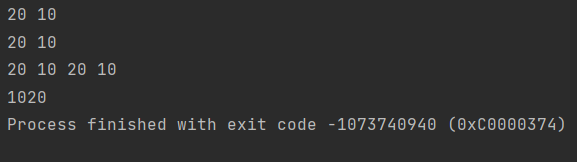
#include <iostream>  
  
using namespace std;  
class Queue{  
private:  
 int\* q;  
 int n;  
public:  
 Queue()  
 {  
 //cout<<"built\n";  
 this->q = NULL;  
 this->n=0;  
 }  
 ~Queue()  
 {  
 //cout<<"dead\n";  
 if(this->q)  
 delete[] this->q;  
 this->n=0;  
 }  
 void operator<<(int x){  
 int \*aux = new int[this->n+1];  
 aux[0]=x;  
 for(int i=1;i<this->n+1;i++)  
 aux[i]=this->q[i-1];  
 delete[] this->q;  
 this->q=aux;  
 this->n++;  
 }  
 void operator>>(int &x){  
 if(this->n==0)return;  
 this->n--;  
 x=this->q[this->n];  
 int \*aux = new int[this->n];  
 for(int i=0;i<this->n;i++)  
 aux[i]=this->q[i];  
 delete[] this->q;  
 this->q=aux;  
 }  
 Queue operator=(Queue &b){  
 delete this->q;  
 this->n = b.len();  
 this->q = new int[this->n];  
 for(int i=0;i<b.len();i++)  
 {  
 this->q[i]=b[i];  
 }  
 return b;  
 }  
 Queue operator+(Queue &b){  
 Queue s = Queue();  
 for(int i=this->n-1;i>=0;i--)  
 {  
 s<<this->q[i];  
 }  
 for(int i=b.len()-1;i>=0;i--)  
 {  
 s<<b[i];  
 }  
 return s;  
 }  
 int operator[](int i){  
 return this->q[i];  
 }  
 void show()  
 {  
 for(int i=0;i<this->n;i++)  
 cout<<this->q[i]<<" ";  
 cout<<endl;  
 }  
 int len()  
 {  
 return this->n;  
 }  
 int norma()  
 {  
 int s=0;  
 for(int i=0;i<this->n;i++)  
 s+=this->q[i];  
 return s;  
 }  
 friend bool operator==(Queue &a,Queue &b);  
 friend bool operator!=(Queue &a,Queue &b);  
 friend bool operator<(Queue &a,Queue &b);  
 friend bool operator>(Queue &a,Queue &b);  
};  
bool operator==(Queue &a,Queue &b){  
 if(a.n!=b.n)return false;  
 for(int i=0;i<a.n;i++)  
 {  
 if(a.q[i]!=b.q[i])return false;  
 }  
 return true;  
}  
bool operator!=(Queue &a,Queue &b){  
 if(a==b)return false;  
 else  
 return true;  
}  
bool operator>(Queue &a,Queue &b){  
 return a.norma()>b.norma();  
}  
bool operator<(Queue &a,Queue &b){  
 return a.norma()>b.norma();  
}  
int main()  
{  
 Queue a = Queue();  
 a<<10;  
 a<<20;  
 int b;  
 Queue d = a;  
 d.show();  
 a.show();  
 (a+d).show();  
 a>>b;  
 cout<<b;  
 a>>b;  
 cout<<b;  
 return 0;  
}

**Screen-uri de la compilarea programului**

1. Timpul, care conţine cîmpurile: ore, minute, secunde:

****

1. Coada, fiind utilizată memoria dinamică :



Concluzie:

* In rezultatul elaborarii lucrarii date s-a pus baza aplicarii in practica a cunostintelor teoretice referitoare la utilizarea operatorilor.
* Astfel se poate judeca despre posibilitatile largi acordate de limbajul C++ referitor la manipularea memoriei in directia dorita de utilizator.

La utilizarea metodelor special predestinate atribuirii nu se poate de garantat utilizarea lor în toate cazurile, înafară de aceasta contrazicem ideea că clasele trebuie să se comporte ca şi tipurile predefinite de date. De asemenea pot apare ambiguităţi, deoarece se va utiliza operatorul de atribuire pentru lucrul cu pointerii şi funcţia pentru utilizarea obiectelor însuşi. Operatorul este o funcţie cu un nume predefinit şi o sintaxă specială la apelare. Operatorii se definesc cu ajutorul cuvîntului cheie operator, urmat de simbolul operaţiei. În rest, este o funcţie obişnuită, care are parametri şi returnează un rezultat. Toţi operatorii se divizează în două grupuri: unari şi binari. Operatorii unari se numesc operatorii cu un singur operand. De ex., operatorul & - de referenţiere, ++ - de incrementare, ! – de negare ş. a. Operatorii binari au doi operanzi: + - adunarea, \* - înmulţirea ş. a. Limbajul C++ permite supraîncărcarea numai a operatorilor existenţi în limbaj. Dintre aceştia nu pot fi supraîncărcaţi operatorii: “.”, “.\*”, “?:”, “::”, “sizeof”, “#”, “##”. Însă pot fi supraîncărcaţi operatorii: “->”, “[]”, “()”, “new” и “new[]”, “delete” и “delete[]” ş. a.