Wstęp do programowania w języku C

Marek Piotrów - Wykład 15
Nowe elementy w C++: przeciążanie operatorów i wzorce klas

3 lutego 2022

Klasa Ulamek I

```
#include <iostream>

using namespace std;

class Ulamek {
   long long licznik;
   long long mianownik;
private:
   long long nwd(long long n,long long m)
   {
      for (long long r; m != 0; n=m,m=r) r=n%m;
      return n;
   }
   void skroc(void) {
      if (mianownik < 0) { licznik=-licznik; mianownik=-mianownik; }
      long long int k = nwd((licznik >= 0 ? licznik : -licznik),mianownik);
      if (k > 1) {
         licznik/=k;
         mianownik/=k;
      }
   }
public:
   Ulamek(long long n=0,long long m=1):licznik(n),mianownik(m) {
      skroc();
   }
   // Konstruktor kopiujący - bedzie wygenerowany automatycznie
   // Ulamek(const Ulamek &u):licznik(u.licznik),mianownik(u.mianownik) { }
```

Klasa `Ulamek` II

```
bool operator== (const Ulamek &u) const {
   return licznik==u.licznik && mianownik==u.mianownik;
}
Ulamek operator+ (const Ulamek &u) const {
   return Ulamek(licznik*u.mianownik+mianownik*u.licznik,mianownik*u.mianownik);
}
Ulamek operator- (const Ulamek &u) const {
   return Ulamek(licznik*u.mianownik-mianownik*u.licznik,mianownik*u.mianownik);
}
Ulamek operator* (const Ulamek &u) const {
   return Ulamek(licznik*u.licznik,mianownik*u.mianownik);
}
Ulamek operator/ (const Ulamek &u) const {
   return Ulamek(licznik*u.mianownik,mianownik*u.licznik);
}
double value(void) const {
   return static_cast<double>(licznik)/mianownik;
}

friend ostream & operator<< (ostream &os,Ulamek u) {
   os<<u.licznik;
   if (u.mianownik != 1) os<<'/'<<u.mianownik;
   return os;
}
friend istream & operator>> (istream &is,Ulamek &u) {
   long long n,m;
   char c;
```

Klasa `Ulamek` III

```
        is>>n>>c>>m;
        u=Ulamek(n,m);
        return is;
    }
};
```

Program testujący klasę

```
#include <iostream>
#include <iomanip>
#include "ulamek.h"

using namespace std;


Ulamek pi_drugich(int n,int a)
{ // wzor Newtona Pi/2 = 1 + 1/3 * (1 + 2/5 * (1 + 3/7 * (1 + 4/9 * ( ... ))))
   if (n > 0)
      return Ulamek(1)+Ulamek(a,2*a+1)*pi_drugich(n-1,a+1);
   else
      return Ulamek(3,2);
}

int main(void)
{
   Ulamek u1,u2,u3;
   Ulamek pi=pi_drugich(22,1)*Ulamek(2);

   cin>>u1>>u2; u3=u1/u2;
   cout<<u1<<" + "<<u2<<" = "<<u1+u2<<endl;
   cout<<u1<<" - "<<u2<<" = "<<u1-u2<<endl;
   cout<<u1<<" * "<<u2<<" = "<<u1*u2<<endl;
   cout<<u1<<" / "<<u2<<" = "<<u3<<endl;

   cout<<"Pi = "<<pi<<" = "<<setprecision(10)<<pi.value()<<endl;
   return 0;
}
```

Wzorzec klasy kolejka

```
#include <cstddef>
using namespace std;

template <class Typ> class kolejka
{
private:
   class element
   {
   friend class kolejka;
   private:
      Typ info;
      element *nastepny;
   public:
      element(element *&list, const Typ &i);
      ~element(void);
   public:
      const Typ wartosc(void) { return(info); }
   };
   element *lista;
public:
   kolejka(void);
   ~kolejka(void);
public:
   bool pusta(void) { return lista==NULL; }
   void wstaw(const Typ &i);
   void usun(void);
   const Typ podaj(void);
};
```

Kolejka - konstruktory i destruktory

```
template <class Typ>
kolejka<Typ>::element::element(element *&list, const Typ &i):info(i)
{
   if (list != NULL) {
      nastepny=list->nastepny;
      list->nastepny=this;
   }
   else nastepny=this;
   list=this;
}

template <class Typ>
kolejka<Typ>::element::~element(void)
{

}

template <class Typ>
kolejka<Typ>::kolejka(void)
{
   lista=NULL;
}

template <class Typ>
kolejka<Typ>::~kolejka(void)
{
   while (lista != NULL) usun();
}
```

Kolejka - operacje

```
template <class Typ>
void kolejka<Typ>::wstaw(const Typ &i)
{
   new element(lista,i);
}

template <class Typ>
void kolejka<Typ>::usun(void)
{
   if (lista == NULL)
      return;
   if (lista->nastepny == lista)
   {
      delete lista;
      lista=NULL;
      return;
   }
   element *pom;
   pom=lista->nastepny;
   lista->nastepny=pom->nastepny;
   delete pom;
}

template <class Typ>
const Typ kolejka<Typ>::podaj(void)
{
   if (lista == NULL) throw 2;
   return lista->nastepny->wartosc();
}
```

Kolejka - test wzorca klasy

```
#include <iostream>
#include <string>
#include "kolejka.h"

using namespace std;

int main(void)
{
   kolejka<int> kint;
   kolejka<string> kstring;

   cout<<"=== Podaj liczby do kolejki (0-koniec) ==="<<endl;
   for (int liczba; cin>>liczba,liczba != 0; ) kint.wstaw(liczba);
   cout<<"=== Zawartosc kolejki liczb ==="<<endl;
   for ( ; !kint.pusta(); kint.usun()) cout<<kint.podaj()<<' ';
   cin.ignore(128,'\n');

   cout<<endl<<"=== Podaj napisy do kolejki (pusty-koniec) ==="<<endl;
   for (char buf[128]; cin.getline(buf,sizeof(buf)),buf[0] != 0; )
      kstring.wstaw(string(buf));
   cout<<"=== Zawartosc kolejki napisow ==="<<endl;
   for ( ; !kstring.pusta(); kstring.usun())
     cout<<'\"'<<kstring.podaj()<<"\" ";
   cout<<endl<<"====== Koniec ========"<<endl;
   return 0;
}
```

Kolejka - test wzorców klas z biblioteki STL

```
#include <iostream>
#include <string>
#include <list>
#include <queue>

using namespace std;

int main()
{
   list<int> kint;
   queue<string> kstring;
   queue<pair<int,string> > pary;

   cout<<"=== Podaj liczby do listy (0-koniec) ==="<<endl;
   for (int liczba; cin>>liczba,liczba != 0; ) kint.push_back(liczba);
   cout<<"=== Zawartosc listy liczb ==="<<endl;
   for ( ; !kint.empty(); kint.pop_front()) cout<<kint.front()<<' ';
   cin.ignore(128,'\n');

   cout<<endl<<"=== Podaj napisy do kolejki (pusty-koniec) ==="<<endl;
   for (char buf[128]; cin.getline(buf,sizeof(buf)),buf[0] != 0; )
      kstring.push(string(buf));
   cout<<"=== Zawartosc kolejki napisow ==="<<endl;
   for (int i=1 ; !kstring.empty(); i++,kstring.pop()) {
     pary.push(pair<int,string>(i,kstring.front()));
     cout<<'\"'<<pary.back().second<<"\" ";
   }
   cout<<endl<<"====== Koniec ========"<<endl;
}
```

Dziękuję za uwagę

Dziękuję wszystkim za uwagę.