Język C++ zajęcia nr 11

Część I – Funkcje operatorowe

Stosowanie wygodnej notacji operatorowej (a nie tylko funkcyjnej) do działań na obiektach klas jest możliwe po odpowiednim przeciążeniu istniejących w języku C++ operatorów. Służą temu funkcje operatorowe definiujące działanie dostępnych operatorów. Działanie tych operatorów jest zdefiniowane na stałe dla standardowych typów języka. Można jednak dokonać ich przeciążenia pod warunkiem, że jednym lub obydwoma argumentami operatora jest obiekt pewnej klasy. Warunek ten nie jest wymagany przy przeciążaniu operatorów **`new`** i **`delete`**.

Nie podlegają przeciążeniu operatory:	**.**	**.***	**?:**	**::**	***sizeof***

Funkcja operatorowa jest funkcją o nazwie, którą symbolicznie przedstawia zapis:

operator@

gdzie symbol @ reprezentuje jeden z operatorów występujących w języku C++. Przykładowe nazwy funkcji operatorowych to:

```
operator+

operator*

operator( )

operator>>
```

Przeciążenie operatora nie można zmieniać jego:

- **składni**: nie można definiować nowych symboli operatorów np. \ lub @, ani zmieniać formy zapisu istniejących np. operator || zapisywać w postaci |x|,
- **argumentowości**: operatory jednoargumentowe po przeciążeniu muszą pozostać również jednoargumentowymi, to samo dotyczy operatorów dwuargumentowych,
- **priorytetu**: po przeciążeniu operatory zachowują swoje ustalone miejsce w hierarchii priorytetów operatorów języka C++,
- **łączności**: operatory o łączności lewostronnej po przeciążeniu pozostają lewostronnymi, analogicznie zachowują się operatory o łączności prawostronnej.

Definiowanie funkcji operatorowej

Definiowanie funkcji operatorowej przebiega podobnie jak w przypadku zwykłej funkcji. Forma definicji zależy od tego, czy funkcja jest, czy nie jest składową pewnej klasy. Funkcja operatorowa, której argumentami są obiekty dwóch różnych klas, może być zadeklarowana jako:

- funkcja globalna nieskładowa,

- funkcja składowa jednej klasy,

- funkcja składowa drugiej klasy.

Przypadek pierwszy. **Funkcja operatorowa NIESKŁADOWA.** Deklaracja może mieć jedną z następujących postaci:

typR operator@ (typ1, typ2) ;

typR operator@ (typ1) ;

gdzie:

typR jest typem zwracanego rezultatu funkcji,

typ1 i ***typ2*** są typami argumentów.

Pierwsza postać dotyczy operatorów dwuargumentowych (**przynajmniej jeden z argumentów musi być typu obiektowego**), druga dotyczy operatorów jednoargumentowych (**argument musi być typu obiektowego**).

Przypadek drugi. **Funkcja operatorowa SKŁADOWA.** Deklaracja może mieć jedną z następujących postaci:

typR operator@ (typ2) ;

typR operator@ (void) ;

gdzie:

typR jest typem zwracanego rezultatu funkcji,

typ2 jest typem argumentu.

Pierwsza postać dotyczy operatorów dwuargumentowych, druga dotyczy operatorów jednoargumentowych. W obydwu przypadkach w deklaracji nie występuje **lewy** argument operatora, którym automatycznie jest **obiekt, na rzecz którego aktywowano funkcję operatorową**. Dla operatora dwuargumentowego podawany jest jedynie jego prawy argument, a dla operatora jednoargumentowego funkcja nie wymaga argumentów.

ZINTERPRETUJ POSZCZEGÓLNE ELEMENTY KODU ŹRÓDŁOWEGO!!!

Klasa z funkcjami operatorowymi:

```
#include <iostream>

using namespace std;

class complex
{
  public:
    float re_part;
    float im_part;
  //-------------------- Konstruktory
    complex(){re_part=0; im_part=0;}
    complex(float re, float im=0) {re_part=re; im_part=im;}
  //-------------------- Deklaracje składowych funkcji operatorowych
    complex operator+ (complex& arg2);
    int operator< (complex& arg2);
};

//-------------------- Definicje składowych funkcji operatorowych

complex complex::operator+ (complex& arg2)
{
  complex sum;            // Pomocniczy obiekt lokalny typu  complex
  sum.re_part = re_part + arg2.re_part;
  sum.im_part = im_part + arg2.im_part;
  return sum;
}

int complex::operator< (complex& arg2)
{
  float md1, md2;
  md1 = re_part*re_part + im_part*im_part;
  md2 = arg2.re_part*arg2.re_part + arg2.im_part*arg2.im_part;
  return md1 < md2;
}
```

```
//-------------------- Definicje globalnych funkcji operatorowych
complex operator- (complex& arg1, complex& arg2)
{
  complex dif;
  dif.re_part = arg1.re_part - arg2.re_part;
  dif.im_part = arg1.im_part - arg2.im_part;
  return dif;
}

int operator> (complex& arg1, complex& arg2)
{
  float md1, md2;
  md1 = arg1.re_part*arg1.re_part + arg1.im_part*arg1.im_part;
  md2 = arg2.re_part*arg2.re_part + arg2.im_part*arg2.im_part;
  return md1 > md2;
}

ostream& operator<< (ostream& out,complex& c)
{
    out<<"("<<c.re_part<<","<<c.im_part<<")";
    return out;
}

int main()
{
    complex a;
    complex b=5;
    complex c(1,2);
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    complex d = b+c+b;
    cout << "d = " << d << endl;
    complex e = b-c;
    cout << "e = " << e << endl;
    if(b<c) cout << "liczba b jest mniejsza od c" << endl;
    else cout << "liczba b nie jest mniejsza od c" << endl;
}
```

Oczekiwane wyniki:

```
a = (0,0)
b = (5,0)
c = (1,2)
d = (11,2)
e = (4,-2)
liczba b nie jest mniejsza od c
```

Uwaga!

Operatory inkrementacji i dekrementacji występują w formie przedrostkowej (**++x, --x**) oraz przyrostkowej (**x--, x++**). Rozróżnienie tych form po przeciążeniu operatorów wynika z zasady, że:

- składowa funkcja operatorowa **operator++** bez argumentów lub nieskładowa funkcja operatorowa **operator++** z jednym argumentem odnosi się do operacji preinkrementacji,
- składowa funkcja operatorowa **operator++** z jednym argumentem typu **int** lub nieskładowa funkcja operatorowa **operator++** z dwoma argumentami (jeden typu obiektowego, drugi typu **int**) odnosi się do operacji postinkrementacji.

Analogiczna zasada dotyczy formy przedrostkowej i przyrostkowej przeciążonego operatora dekrementacji.

Część II – Program CPP11 - modyfikacja programu CPP10-1 (operacje na ułamkach) z wykorzystaniem funkcji operatorowych

Otwórz program CPP10-1, zapisz jako CPP11 i dopisz poniższe funkcje:

1. Zadeklaruj i zdefiniuj następujące składowe funkcje operatorowe:

```
Ulamek operator*(int n)
```

Mnoży ułamek przez argument n, zwraca ułamek będący wynikiem operacji.
Uwaga: podczas definiowania lokalnego obiektu pomocniczego (klasy Ulamek) w definicji funkcji wywołaj konstruktor dwuargumentowy, np.: Ulamek pom(1,1); przykładowa definicja tej funkcji:

```
Ulamek Ulamek::operator* (int n)
{
   Ulamek u(1,1);
   u.licznik=licznik*n;
   u.mianownik=mianownik;
   return u;
}
```

```
Ulamek operator+(int n)
```

Dodaje argument n do ułamka, zwraca ułamek będący wynikiem operacji.

```
Ulamek operator-(int n)
```

Odejmuje argument n do ułamka, zwraca ułamek będący wynikiem operacji.

```
Ulamek operator-(void)
```

Zmienia znak bieżącego ułamka, zwraca ułamek będący wynikiem operacji.

2. Zdefiniuj następujące nieskładowe funkcje operatorowe:

```
Ulamek operator*(Ulamek a, Ulamek b)
```

Mnoży ułamki a i b, zwraca ułamek stanowiący ich iloczyn.

```
Ulamek operator+(Ulamek a, Ulamek b)
```

Dodaje ułamki a i b, zwraca ułamek stanowiący ich sumę.

```
int operator<(Ulamek a, Ulamek b)
```

Porównuje ułamki a i b, zwraca 1 gdy a<b, 0 gdy a>=b.

```
ostream& operator<< (ostream& out, Ulamek a)
```

Drukuje zapis ułamka w postaci licznik/mianownik (np. 3/5).

3. (*zadanie dodatkowe - nadobowiązkowe***) Zdefiniuj składową funkcje operatorową:**

```
void operator!(void)
```

> ***Upraszcza ułamek*** *do najprostszej postaci (licznik i mianownik – względnie pierwsze) i wynik umieszcza w bieżącym ułamku.* ***Wskazówka: sprawdź wszystkie dzielniki mianownika (2<=dzielnik<=mianownik) i jeśli zarówno licznik jak i mianownik są podzielne przez dany dzielnik to wykonaj takie dzielenia.***

4. Zmodyfikuj oba konstruktory tak, aby wykonywały kontrolę, czy mianownik jest większy od zera. *Przerwanie programu (np. w przypadku błędu) umożliwia funkcja exit() – prototyp w pliku stdlib.h*

5. Napisz funkcję main() która:

- *tworzy dwa ułamki x i y: 6/21 oraz drugi wprowadzony z klawiatury,*
- *mnoży ułamek y przez 5 (przypisując wynik za y),*
- *drukuje wartość ułamków x i y za pomocą funkcji drukuj,*
- *porównuje ułamki x i y, wypisuje komunikat stanowiący rezultat porównania,*
- *upraszcza ułamki x i y do najprostszej postaci (o ile zdefiniowano operator !)*
- *drukuje (przy użyciu operatorów strumieniowych <<) w jednej linii ułamki x i y w postaci licznik/mianownik, przechodzi do nowej linii,*
- *drukuje (w postaci jak wyżej) y, y+1 , y-1 , oraz –y*
- *drukuje: x, y, x*y, x+y, -(x*y)*
- *przypisuje za y ułamek –(x*y), upraszcza y (o ile istnieje operator !), drukuje x, y,*
- *ew. wykonuje inne czynności realizowane przez utworzone funkcje.*

Kompletny program CPP11 (kod źródłowy wraz z oknem wyników) należy przesłać za pośrednictwem Moodle w wyznaczonym terminie.