1. Wprowadzenie do przeciążania operatorów

Oprócz słów kluczowych i typów, operatory stanowią podstawowy element każdego języka programowania wysokiego poziomu (Python, Haskel czy C++). Korzystając z języka programowania C++ użytkownik ma do dyspozycji typy wbudowane (int, float, unsigned, itd.), jak również może utworzyć własne typy przechowujące różne typy danych (struktury, klasy). O ile w specyfikacji standardu C++ zdefiniowano zestaw operatorów, o tyle tworzenie nowych operatorów w przeciwieństwie do nowych typów nie jest możliwe. Istnieje jednak możliwość przeciążania operatorów już istniejących dla czynności realizowanych na obiektach typu klasa, co umożliwia w razie potrzeby odmienne od typowego działanie operatorów. Operatory można zatem przeciążać, podobnie jak inne funkcje i metody języka C++.

Przeciążanie operatorów (ang. *operator overloading*), nazywane zamiennie również **przeładowaniem**, pozwala nadawać operatorom nowe znaczenie, dzięki czemu mogą być one wykorzystane w odniesieniu do obiektów zdefiniowanych klas (implementacja klasy przechowującej złożony typ danych z przeciążonymi operatorami).

Przeciążania operatora można dokonać na dwa sposoby. Pierwszym jest wykorzystanie niezależnej funkcji (na ogół zaprzyjaźnionej z jedną lub większą liczbą klas). Przykładowo dla operatora dwuargumentowego o nazwie „przeciazenie” otrzymujemy:

x przeciazenie y

Wyrażenie to jest tożsame z wywołaniem:

operator przeciazenie (x , y).

Drugi ze sposobów opiera się na użyciu funkcji składowej, co odpowiada wywołaniu:

x.operator przeciazenie (y)

Aby funkcja globalna posiadała dostęp do danych prywatnych klasy, funkcja operatorowa musi być zadeklarowana jako funkcja zaprzyjaźniona z klasa np.:

friend nazwa_funkcji *operator* * (liczba_1, liczba_2)

W tym celu konieczne jest umieszczenie prototypu funkcji w definicji klasy, który poprzedzony jest słowem kluczowym *friend.* Jako że do funkcji zaprzyjaźnionych nie jest przekazywany wskaźnik *this*, konieczne staje się przekazanie operandów w sposób jawny, co wymusza przekazanie dwóch argumentów dla funkcji operatorowej dwuargumentowej,

w odróżnieniu od przeładowania za pomocą zwykłej funkcji składowej klasy. Zgodnie ze standardem C++, w sytuacji, gdy operator dwuargumentowy przeładowany jest za pomocą funkcji składowej klasy, jawnie należy przekazać do niego tylko jeden argument. Drugi z argumentów przekazywany jest w sposób niejawny poprzez wskaźnik *this*.

Operatory przeciąża się poprzez uprzednie zdefiniowanie funkcji operatorowej. Jako argumentów operatorów używa się typów zadeklarowanych zewnętrznie (class, enum, struct, union). Istnieje zasada, że obiekt, który spowodował wywołanie funkcji operatora zawsze umiejscowiony jest po lewej stronie operacji, zaś obiekt występujący po prawej stronie przekazywany jest do funkcji w postaci argumentu.

W przypadku, gdy operator modyfikuje operandy należy go zdefiniować jako funkcję składową klasy. Jako przykład można podać operatory: „ = ” , „ += ”, „ -= ”, „ ++ ”, itp. W innym przypadku operator powinien być zdefiniowany jako funkcja globalna lub zaprzyjaźniona. Jako przykład można podać m.in. operatory: „ + ”, „ - ”, ” == ”.
Przykładowa implementacja przeciążenia operatora mnożenia liczb ułamkowych z użyciem operatorowej funkcji składowej klasy ma postać:

```
ulamek ulamek :: operator * (ulamek a)
{ulamek wynik;
wynik.licznik = licznik * a.licznik;
wynik.mianownik = mianownik * a.mianownik;
return wynik;}
```

1.1 Wykaz operatorów możliwych do przeładowania

W ramach realizacji niniejszej instrukcji laboratoryjnej studenci zapoznają się z mechanizmem przeciążania operatorów arytmetycznych, przypisania i porównania. Ponadto celem jest realizacja przeciążania operatorów strumienia wejścia >> i wyjścia <<.

– >	Operator dostepu (Structure memeber pointer reference)
New	Dynamicznie alokowana pamięć (Dynamically allocate memory)
Delete	Dynamicznie usuwana pamięć (Dynamically deallocated memory)
++	Inkrementacja (Increment)
–	Dekrementacja (Decrement)
-	Minus (Unary minus)
!	Logiczna negacja (Logical negation)
~	Dopełnienie logiczne (One's complement)
*	Dereferencja (Indirection)
*	Mnożenie (Multiplication)
/	Dzielenie (Division)
%	Modulo (Modulus (remainder))
+	Dodawanie (Addition)
-	Odejmowanie (Subtraction)
<<	Przesunięcie (Left shift)
>>	Przesunięcie (Right shift)
<	Mniej niż (Less than)
<=	Mniej niz lub rowne (Less than or equel to)
>	Większe niż (Greater than)
>=	Większe niż lub równe (Greater than or equel to)
==	Równe (Equal to)
!=	Różne (Not equal to)
&&	Logiczne AND (Logical AND)
||	Logiczne OR (Logical OR)
&	Bitowe AND (Bitwise AND)
^	Bitowe XOR (Bitwise exlusive OR)
|	Bitowe OR (Bitwise inclusive OR)
=	Przypisanie (assignment)
+= -= *=	Przypisanie (assignment)
/= %= &=	Przypisanie (assignment)
^= | =	Przypisanie (assignment)
<<=>>=	Przypisanie (assignment)

Tab. 1. Wykaz operatorów możliwych do przeładowania

[Źródło: P. Mikołajczyk, „Język C++ Podstawy Programowania", Lublin 2011, s. 145]

Oprócz wskazanych powyżej operatorów przeciążyć jeszcze można operator wywołania funkcji (ang. *Function call*): „ () "; operator elementu tablicy (ang. *Array element*): „ [] " oraz operator przecinka (ang. *Comma*): „ , ". W ten sposób, przedstawiony w instrukcji zbiór operatorów do przeciążenia jest zbiorem kompletnym.

O ile każdy z wymienionych wyżej operatorów można przeciążyć (przeładować) w dowolny sposób, tak aby realizował inną operację, o tyle istnieją pewne pisane reguły

mówiące o tym, że operator jednoargumentowy musi pozostać operatorem unarnym (związanym z jednym operandem), zaś operator dwuargumentowy – operatorem binarnym (związanym z dwoma operandami). W praktyce nie wszystkie operatory można przeciążyć. Jako przykład można podać: operator kropki „ . ”, operator zasięgu „ :: ”, operator wyłuskania wskaźnika do składowej „ .* ”, operator rozmiaru „ sizeof ” czy operator warunkowy „ ?: ”. Poza tym standard C++ uniemożliwia tworzenie nowych symboli operatorów, modyfikowanie łączności i kolejności operatorów. Ponadto operator musi być zdefiniowany w taki sposób, aby co najmniej jeden członek klasy stanowił jego operand, albo też operator był członkiem klasy.

1.2 Przeciążanie operatora <<

Załóżmy, że mamy do dyspozycji klasę *Punkt* składającą się z trzech prywatnych współrzędnych typu *int*. Współrzędne punktu można zainicjować przy pomocy odpowiedniego konstruktora lub przy użyciu klawiatury, wykorzystując w tym celu metodę *wprowadz*(). Natomiast wyświetlić współrzędne na ekranie można przy pomocy metody *wyswietl*(). Aby tego dokonać wykorzystać należy między innymi operatory strumienia wejściowego oraz wyjściowego. Oto przykład:

```
#include <iostream>

using namespace std;

class Punkt{
        int x,y,z;
public:
        Punkt (int wart_x=0, int wart_y=0, int wart_z=0):
        x(wart_x), y(wart_y), z(wart_z){
        cout<<"Punkt (0, 0, 0) został powołany do życia"<<endl;
        }

        void wprowadz (){
        cout<<"Wprowadz wspolrzedne punktu: x y z"<<endl;
```

```
        cin>>x>>y>>z;
        }

        void wyswietl(){
        cout<<"Oto wsoplrzedne punktu (x, y, z)"<<endl;
        cout<<"("<<x<<", "<<y<<", "<<z<<")"<<endl;
        }
};

int main(void)
{
        Punkt p;
        p.wprowadz();
        p.wyswietl();
        return 0;
}
```

Alternatywny sposób wyprowadzania wartości współrzędnych na ekran polega na przeciążeniu operatora strumienia wyjściowego << (który jest de facto przeciążonym operatorem przesunięcia bitowego w lewo), aby odpowiednio wykorzystać możliwości klasy *ostream* do wyświetlenia składowych punktu na ekranie komputera. Poniżej zamieszczono przykład ilustrujący przeciążenie operatora << w celu wyświetlenia współrzędnych punktu na ekranie komputera:

```
#include <iostream>

using namespace std;

class Punkt{
        int x,y,z;
public:
        Punkt (int wart_x=0, int wart_y=0, int wart_z=0):
        x(wart_x), y(wart_y), z(wart_z){
        cout<<"Punkt (0, 0, 0) został powołany do życia"<<endl;
```

```
        }

        void wprowadz (){
        cout<<"Wprowadz wspolrzedne punktu: x y z"<<endl;
        cin>>x>>y>>z;
        }

        friend ostream & operator<<(ostream & wyjscie , Punkt & p){
        wyjscie<<"Oto wspolrzedne punktu (x, y, z)"<<endl;
        wyjscie<<"("<<p.x<<", "<<p.y<<", "<<p.z<<")"<<endl;
        return wyjscie;
        }
};

int main(void)
{
        Punkt p;
        p.wprowadz();
        cout<<p;
        return 0;
}
```

1.3 Przeciążanie operatora >>

Przeładowanie operatora strumienia wejściowego >> (który jest de facto przeciążonym operatorem przesunięcia bitowego w prawo), w celu wprowadzenia współrzędnych punktu p z klawiatury, odbywa się analogicznie jak w przypadku przeciążania operatora operatora <<, z tą różnicą, że tym razem będziemy wykorzystywać klasę *istream*. Oto przykład ilustrujący takie właśnie przeciążenie:

```
#include <iostream>

using namespace std;
```

```
class Punkt{
        int x,y,z;
public:
        Punkt (int wart_x=0, int wart_y=0, int wart_z=0):
        x(wart_x), y(wart_y), z(wart_z){
        cout<<"Punkt (0, 0, 0) został powołany do życia"<<endl;
        }

        friend istream & operator>>(istream & wejscie , Punkt & p){
        wejscie>>p.x>>p.y>>p.z;
        return wejscie;
        }

        friend ostream & operator<<(ostream & wyjscie , Punkt & p){
        wyjscie<<"Oto wspolrzedne punktu (x, y, z)"<<endl;
        wyjscie<<"("<<p.x<<", "<<p.y<<", "<<p.z<<")"<<endl;
        return wyjscie;
        }
};

int main(void)
{
        Punkt p;
        cout<<"Wprowadz wspolrzedne punktu: x y z"<<endl;
        cin>>p;
        cout<<p;
        return 0;
}
```

Z przykładów zamieszczonych w niniejszej instrukcji wynika elastyczność języka C++, który umożliwia dość znaczną swobodę w przeciążaniu dozwolonych operatorów. Ograniczeniem tej swobody jest inwencja programisty oraz czytelność kodu, która przykładowo nie powinna dopuszczać przeciążania operatora dodawania tak, żeby za jego pomocą wykonywane było odejmowanie.

2. Zadania do wykonania

1. Napisać program z wykorzystaniem dwóch funkcji operatora jako funkcji składowych klasy, będących odpowiednio przeciążonym operatorem dodawania i odejmowania. Program powinien umożliwić operacje dodawania i odejmowania składowych dwóch wektorów dwuwymiarowych. Jako wynik użytkownik powinien uzyskać dwa nowe wektory o współrzędnych wynikających odpowiednio z dodawania i odejmowania składowych poszczególnych wektorów.
2. Napisać program z wykorzystaniem funkcji operatora jako funkcji składowej klasy, będącej przeciążonym operatorem mnożenia. Program powinien umożliwić mnożenie dwóch dowolnych ułamków oraz zwracać ułamek będący iloczynem tych ułamków.
3. Zrealizować zadanie 2 z wykorzystaniem przeciążonego operatora mnożenia jako niezależnej operatorowej funkcji globalnej (zaprzyjaźnionej z klasą).
4. Napisać program służący do porównywania dwóch wektorów o trzech składowych (x, y, z) z wykorzystaniem mechanizmu przeciążania operatorów „ == ” oraz „ != ”. Program powinien zrealizować przeciążenia jako funkcje składowe danej klasy.
5. Napisać analogiczny program do zadania 4 z wykorzystaniem przeciążeń jako niezależnych operatorowych funkcji globalnych (zaprzyjaźnionych z daną klasą).
6. Napisać program z wykorzystaniem operatora przypisania „ = ”, wyświetlający informacje na temat czasu (godzina, minuta, sekunda) z wykorzystaniem klasy nie posiadającej operatorowej funkcji przypisania (przypisanie danych składowych ma nastąpić z wykorzystaniem przeciążenia automatycznie wygenerowanego przez kompilator – *memberwise assignment*).
7. Napisać analogiczny program do zadania 6 z wykorzystaniem przeciążonego operatora przypisania „ = ” (przeciążenie jako funkcja składowa danej klasy).

   UWAGA! Ponieważ funkcja operatorowa przypisania nie może być funkcją zaprzyjaźnioną, musi stanowić funkcję składową danej klasy.
8. Uzupełnić dowolny program od 1 do 5 tak, żeby zaprezentować przeciążanie operatorów strumienia wejściowego i wyjściowego.