

Techniki Efektywnego Programowania – zadanie 2 Przeciążanie operatorów

Dlaczego wprowadzono przeciążanie operatorów?

Dla człowieka zapis $2+3=5$ jest intuicyjny i wygodny. Wyobraźmy sobie, że mamy klasę, której obiekty implementują liczby (np. bardzo duże liczby, które mogą mieć nawet kilka tysięcy cyfr). Podstawowymi operacjami, jakie będziemy wykonywać na obiektach tej klasy będą dodawanie, odejmowanie, dzielenie, mnożenie i operacja przypisania wartości. Deklaracja tej klasy może wyglądać na przykład tak:

```
#define NUMBER_DEFAULT_LENGTH 10
class CNumber
{
public:
    CNumber() {i_length = NUMBER_DEFAULT_LENGTH; pi_number = new
int[i_length];};
    ~CNumber() {delete pi_number;}

    void vSet(int iNewVal);
    void vSet(CNumber &pcNewVal);
    CNumber vAdd(CNumber &pcNewVal);
    CNumber vSub(CNumber &pcNewVal);
    CNumber vMul(CNumber &pcNewVal);
    CNumber vDiv(int iNewVal);
    CNumber vAdd(int iNewVal);
    CNumber vSub(int iNewVal);
    CNumber vMul(int iNewVal);
    CNumber vDiv(int iNewVal);
private:
    int *pi_number;
    int i_length;
} //class CNumber
```

Zmiast wskaźnika, używana jest referencja (symbol &), która oznacza to samo, ale **w tym przypadku** upraszcza zapis. Będzie to widoczne na dalszych przykładach.

Proszę zauważyć, że zgodnie z definicją, metody `vAdd`, `vSub`, `vMul`, `vDiv` nie modyfikują obiektu na rzecz, którego zostały wywołane, tylko zwracają wynik działania. Może się to wydawać wątpliwe z punktu widzenia samej idei programowania obiektowego. Jeżeli programujemy obiektowo, to oczekujemy, że dana metoda będzie modyfikować pola obiektu na rzecz którego następuje wywołanie. Czyli można powiedzieć, że na przykład metoda dodająca powinna wyglądać tak:

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

```
void vAdd(CNumber &pcNewVal);
```

a jej użycie tak:

```
CNumber c_value, c_add;  
/* initialize c_value, c_add */  
c_value.vAdd(c_add);
```

Wynik dodawania w powyższym przykładzie byłby akumulowany w obiekcie `c_value`. Jednak takie wykonanie metod obsługujących arytmetykę byłoby niewygodne, dlatego zgodnie z deklaracją klasy `CNumber`, użycie będzie wyglądać tak:

```
CNumber c_result, c_value, c_add;  
/* initialize c_value, c_add */  
c_result.vSet(c_value.cAdd(c_add));
```

A operacja `cAdd`, nie będzie modyfikować obiektu `c_value`, tylko zwracać wynik kopiowany do obiektu `c_result`.

Jeżeli przy użyciu klasy `CNumber` będziemy chcieli wykonać następujące obliczenia:
 $result = (arg / 5) * 2 + arg$, gdzie arg to wartość argumentu wprowadzona przez użytkownika, to będzie to wyglądać tak:

```
CNumber c_result, c_arg;  
/* initialize c_arg */  
c_result.vSet(c_arg.cDiv(5).cMul(2).cAdd(c_arg));
```

Można uznać, że znacznie bardziej czytelny byłby zapis:

```
CNumber c_result, c_arg;  
/* initialize c_arg */  
c_result = (c_arg / 5) * 2 + c_arg;
```

Kolejną zaletą powyższego zapisu jest możliwość stosowania nawiasów i kolejności wykonania działań normalnej dla matematycznego zapisu. Można uznać, że im bardziej skomplikowane działanie, tym bardziej zapis używający nawiasów i operatorów powinien być czytelniejszy od zapisu wywołującego kolejne metody.

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Przeciążanie operatorów w C++

W C++ można przeciążyć następujące operatory:

+ - * / % ^ & | ~ ! , < > <= >= ++ -- << >> == != && || += -= /= *= %= ^= &= |= <<= >>= [] () -> ->* new new[] delete delete[]

Niektóre z nich są 1-argumentowe (np. operator ++), a inne 2-argumentowe (np. operator *).

Deklaracja operatora ma postać:

```
<Typ> operator <symbol> (<parametry>);
```

Przykład dla klasy CNumber:

```
class CNumber
{
public:
    CNumber() {i_length = NUMBER_DEFAULT_LENGTH; pi_number = new
int[i_length];};
    ~CNumber() {delete pi_number;}

    void operator=(CNumber &pcNewVal);
    CNumber operator+ (CNumber &pcNewVal);
    CNumber operator* (CNumber &pcNewVal);
    CNumber operator- (CNumber &pcNewVal);
    CNumber operator/ (CNumber &pcNewVal);
    CNumber operator+ (int iNewVal);
    CNumber operator* (int iNewVal);
    CNumber operator- (int iNewVal);
    CNumber operator/ (int iNewVal);
private:
    int *pi_number;
    int i_length;
} //class CNumber
```

Poszczególne operatory są oprogramowywane jak normalne metody i nimi właśnie są w istocie. Zapis operatorowy, zawsze sprowadza się do serii wywołań poszczególnych metod na rzecz odpowiednich obiektów. Na przykład zapis:

```
c_result = c_arg + 2;
```

Sprowadza się do wywołania:

```
c_result.operator=(c_arg.operator+(2));
```

Szczególnie istotnym operatorem jest operator przypisania (`operator=`). Podobnie jak konstruktor kopiujący, jest on definiowany niejawnie dla każdej klasy. Podobnie, jak w przypadku konstruktora kopiującego działanie domyślnego `operator=`, polega skopiowaniu wartości wszystkich pól obiektu. **Zastanów się, kiedy domyślny `operator=` jest wystarczający, a kiedy niezbędne jest zdefiniowanie własnego `operator=`.**

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Zazwyczaj, przy użyciu operatorów ich argumenty pobierane są jako referencje, a wynik jest zwracany statycznie. Dzięki temu możliwy jest zapis:

```
c_result = c_arg + c_arg + 2;
```

Gdyby zamiast referencji operatory pobierały wskaźniki, to wyglądałoby to tak:

```
class CNumber
{
public:
    //...
    void operator=(CNumber *pcNewVal);
    CNumber operator+(CNumber *pcNewVal);
    CNumber operator+(int iNewVal);
    //...
} //class CNumber
```

Użytkowanie byłoby mniej wygodne, ponieważ, choć referencja jest wskaźnikiem, to w kodzie odwołania do zmiennych będących referencjami są takie same jak do zmiennych będących obiektami. W przypadku ponierania wskaźników, użycie wyglądałoby tak:

```
c_result = &(c_arg + &(c_arg + 2));
```

Uwaga: w przypadku użycia operatorów i zwracania wyniku przez wartość (jak w powyższych przykładach) często niezbędne jest prawidłowe zdefiniowanie konstruktora kopiującego.

Jedną z wad użycia operatorów jest duża liczba tworzonych kopii obiektów. **Szczegółowa analiza jakie obiekty są tworzone, oraz kiedy zostanie przedstawiona na wykładzie.** Można i warto jednak sprawdzić to samodzielnie w ramach ćwiczeń na laboratorium.

Uwaga: Jednym z rozwiązań, które można zastosować w ramach użycia operatorów jest tzw. konstruktor przenoszący (ang. *move constructor*). Ta tematyka zostanie poruszona w ramach listy numer 8, która wprowadza elementy C++11 i C++14.

„ZPR PWR – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Zadanie

UWAGI:

1. Pisząc własny program można użyć innego nazewnictwa niż to przedstawione w treści zadania i w przykładach. Należy jednak użyć jakiejś spójnej konwencji kodowania, zgodnie z wymaganiami kursu.
2. Nie wolno używać wyjątków (jest to jedynie przypomnienie, wynika to wprost z zasad kursu).
3. Wolno używać wyłącznie komend ze standardu C++98

Rozwiń klasę CTable z listy numer 2 o działanie operatorów. Wykonaj poniższe ćwiczenia.

1. Utwórz dwa obiekty klasy CTable, tak jak w przykładzie i wykonaj następującą operację (zdefiniuj wcześniej `operator=` tak jak podano poniżej)

```
void CTable::operator=(CTable &pcOther)
{
    pi_table = pcOther.pi_table;
    i_tab_len = pcOther.i_tab_len;
} //void CTable::operator=(CTable &pcOther)
```

```
CTable c_tab_0, c_tab_1;
c_tab_0.bSetNewSize(6);
c_tab_1.bSetNewSize(4);
c_tab_0 = c_tab_1;
```

Jak skończyło się wykonanie powyższego programu? Dlaczego?

2. Usuń destruktor z klasy CTable. Czy coś się zmieniło? Dlaczego?
3. Napisz metodę `void vSetValueAt(int iOffset, int iNewVal)`, która pozwala wpisywać wartości do tablicy (jeżeli jeszcze nie masz takiej metody). Wpisz wartości np. 1,2,3,4... do `c_tab_0` i 51,52,53,54 do `c_tab_1`. Napisz metodę `vPrint()`; wypisującą tablicę na ekran i wykonaj poniższy program.

```
CTable c_tab_0, c_tab_1;
c_tab_0.bSetNewSize(6);
c_tab_1.bSetNewSize(4);
/* initialize table */
c_tab_0 = c_tab_1;
c_tab_1.vSetValueAt(2,123);
c_tab_0.vPrint();
c_tab_1.vPrint();
```

Jakie napisy zostały wyświetlone na ekranie. Czy umiesz już w pełni wytłumaczyć zachowanie programu w ćwiczeniu nr 1?

4. Oprogramuj operator+ dla klasy CTable, który zwraca konkatencję dwóch tablic.



Fundusze Europejskie
Wiedza Edukacja Rozwój



Politechnika Wrocławska

Unia Europejska
Europejski Fundusz Społeczny



„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Zalecana literatura

Jerzy Grębosz „Symfonia C++”, Wydawnictwo Edition, 2000.

Lub inna zalecana dla przedmiotu.

Binky pointer fun (<https://www.youtube.com/watch?v=5VnDaHBi8dM>)