C++: Przeciążanie operatorów

Zajęcia laboratoryjne nr 1 - Metodyka i Techniki Programowania II
Sem. zimowy, r. akadem. 2021/2022

Autor instrukcji: dr inż. Andrzej Staniszewski

Wstęp

Każdy **podstawowy typ zmiennej** posiada **zbiór wartości**, które zmienne tego typu mogą przyjmować, oraz **operacje**, jakie można przeprowadzać na wartościach tego typu. Np. zmienne **typu int** można dodawać +, odejmować -, mnożyć *, dzielić / (…i inne).
Co robi kompilator, gdy natrafi w wyrażeniu na jakiś operator? Sobie tylko znanymi sposobami oblicza on docelową wartość? Jakie to są sposoby?!
Działanie operatora definiuje pewna funkcja, zwana **funkcją operatorową** (ang. *operator function, metoda operatorowa*). Istnieje wiele takich funkcji, które są wbudowane w kompilator i **działają na typach podstawowych**. Dodawanie, odejmowanie i inne predefiniowane działania na liczbach są dostępne bez żadnych starań z naszej strony. Nie znając szczegółów, intuicyjnie zdajemy sobie sprawę, że gdy kompilator C++ napotka w programie wystąpienie operatora, to na podstawie typu zmiennych, ustala jakie operacje ma przeprowadzić. Oczywiście, są to różne operacje dla różnych typów danych.
Klasa jest definicją nowego **typu** (**typ definiowany**). Zatem możemy zdefiniować nowe operacje przeprowadzane na zmiennych tego typu (klasy). W języku C++ nie wolno definiować nowych symboli operacji. Zatem musimy użyć **symboli operatorów wbudowanych**, nadając im nowe znaczenie (są pewne ograniczenia – cztery operatory wbudowane nie mogą być przeciążane). Proces ten nazywa się **przeciążaniem operatora**. Przeciążenie może nadać operatorowi dowolne znacznie, nie ma tez ograniczeń co do wartości zwracanej przez operator (są pewne ograniczenia – wyjątkiem są operatory new i delete). Nowe znaczenie operatora nie musi być w jakikolwiek sposób związane z jego **znaczeniem wbudowanym**.
Przeciążanie operatora to zmodyfikowanie znaczenia operatora na **potrzeby klasy**.
Zatem **przeciążanie** operatora oznacza konieczność **napisania** dla niego **własnej funkcji operatorowej**. Trzeba podać jej argumenty oraz wartość zwracaną i wypełnić kodem. Słowo kluczowe **operator** służy do oznaczenia przedefiniowania operatora.

Przeciążanie operatora (+) i (-) dla łańcuchów znaków (**napisów**)

Chcemy łączenie dwóch napisów zapisać znakiem (**+**), a usunięcie pewnego znaku z napisu/tekstu oznaczyć operatorem (**-**). (Pomijamy istnienie klasy **string**).

TEKST KONTYNUOWANY NA STRONIE NASTĘPNEJ!

a. Rozwiązanie klasyczne

W rozwiązaniu „klasycznym” zastosowalibyśmy funkcje wykonujące pożądane operacje.
Przeanalizuj, przepisz, skompiluj i wykonaj poniższy program:

```
//przeciazanie operatorow (+) i (-) dla napisów - wersja klasyczna

#include <iostream>
#include <string.h>
using namespace std;

class Tekst{
  public:
    Tekst(const char *);
    void dolacz_tekst(const char *);
    void minus_tekst(const char);
    void pisz_tekst(void);
  private:
                char dane[256];
};

Tekst::Tekst(const char *tks){
                strcpy(dane, tks);
}

void Tekst::dolacz_tekst(const char *tks){
                strcat(dane, tks);
}

void Tekst::minus_tekst(const char znak){
  char tymczas[256];
  int i, j;

  for(i=0, j=0; dane[i]; i++)
                if(dane[i] != znak)
                  tymczas[j++] = dane[i];
  tymczas[j] = '\0';
  strcpy(dane, tymczas);
}

void Tekst::pisz_tekst(void){
  cout << dane << endl;
}

int main(void) {
  Tekst tytul("\"Wygraj z C++\"");
  Tekst lekcja("Przeciazanie operatorow");

  tytul.pisz_tekst();
  tytul.dolacz_tekst(" to najlepszy podrecznik.");
  tytul.pisz_tekst();

  lekcja.pisz_tekst();
  lekcja.minus_tekst('r');
  lekcja.pisz_tekst();
}
```

W powyższym przykładzie zdefiniowano metody **dolacz_tekst()** i **minus_tekst()**, które to funkcje przeprowadziły stosowne operacje.
Jakie wyniki otrzymałeś(-aś)?

b. Przeciążanie operatorów – wersja 1

Przeanalizuj, przepisz, skompiluj i wykonaj poniższy program:

```
//przeciazanie operartorow (+) i (-) dla napisow - wersja 1

#include <iostream>
#include <string.h>
using namespace std;

class Tekst{
  public:
    Tekst(const char *);
    void operator+(const char *);
    void operator-(const char);
    void pisz_tekst(void);
  private:
                char dane[256];
};

Tekst::Tekst(const char *tks){
                strcpy(dane, tks);
}

void Tekst::operator+(const char *tks){
                strcat(dane, tks);
}

void Tekst::operator-(const char znak){
  char tymczas[256];
  int i, j;

  for(i=0, j=0; dane[i]; i++)
                if(dane[i] != znak)
                  tymczas[j++] = dane[i];
  tymczas[j] = '\0';
  strcpy(dane, tymczas);
}

void Tekst::pisz_tekst(void){
  cout << dane << endl;
}

int main(void) {
  Tekst tytul("\"Wygraj z C++\"");
  Tekst lekcja("Przeciazanie operatorow");

  tytul.pisz_tekst();
  tytul.operator+(" to najlepszy podrecznik.");
  tytul.pisz_tekst();

  lekcja.pisz_tekst();
  lekcja.operator-('r');
  lekcja.pisz_tekst();
}
```

Jaki wynik uzyskałeś(-aś) po wykonaniu programu?
Przeanalizuj sposób przedefiniowania (kodowania) operatora za pomocą słowa kluczowego **operator**. Zwróć uwagę na sposób wykonania operatora, gdzie wywołujemy go jako **funkcję** (**operatorową**), np. **lekcja.operator-('r')** .

Powyższy kod możemy zapisać w następujący, krótszy sposób – zwróć uwagę na sposób wywołania operatora:

c. Przeciążanie operatorów – wersja 2

Przeanalizuj, przepisz, skompiluj i wykonaj poniższy program:

```
//przeciazanie operatorow (+) i (-) dla tekstow - wersja 2

#include <iostream>
#include <string.h>
using namespace std;

class Tekst{
  public:
    Tekst(const char *);
    void operator+(const char *);
    void operator-(const char);
    void pisz_tekst(void);
  private:
                char dane[256];
};

Tekst::Tekst(const char *tks){
                strcpy(dane, tks);
}

void Tekst::operator+(const char *tks){
                strcat(dane, tks);
}

void Tekst::operator-(const char znak){
  char tymczas[256];
  int i, j;

  for(i=0, j=0; dane[i]; i++)
                if(dane[i] != znak)
                  tymczas[j++] = dane[i];
  tymczas[j] = '\0';
  strcpy(dane, tymczas);
}

void Tekst::pisz_tekst(void){
  cout << dane << endl;
}

int main(void) {
  Tekst tytul("\"Wygraj z C++\"");
  Tekst lekcja("Przeciazanie operatorow");

  tytul.pisz_tekst();
  tytul + " to najlepszy podrecznik.";
  tytul.pisz_tekst();

  lekcja.pisz_tekst();
  lekcja - 'r';
  lekcja.pisz_tekst();
}
```

Jaki wynik uzyskałeś(-aś) po wykonaniu programu?
Powyższy zapis użycia operatora (np. instrukcja **lekcja - 'r';**) jest poprawny, ale nieco dziwny. Jesteśmy przyzwyczajeni do tego, że wyrażenie zawierające operator (np. +) zapisujemy w postaci: zm = zm1 + zm2;
Zatem modyfikujemy kod tak, aby zachować tę konwencję.

d. Przeciążanie operatorów – wersja 3

Przeanalizuj, przepisz, skompiluj i wykonaj poniższy program:

```
//przeciazanie operatorow (+) i (-) dla tekstow - wersja 3

#include <iostream>
#include <string.h>
using namespace std;

class Tekst{
  public:
    Tekst(const char *);
    char *operator+(const char *);
    char *operator-(const char);
    void pisz_tekst(void);
  private:
                char dane[256];
};

Tekst::Tekst(const char *tks){
                strcpy(dane, tks);
}

char *Tekst::operator+(const char *tks){
                return(strcat(dane, tks));
}

char *Tekst::operator-(const char znak){
  char tymczas[256];
  int i, j;

  for(i=0, j=0; dane[i]; i++)
                if(dane[i] != znak)
                  tymczas[j++] = dane[i];
  tymczas[j] = '\0';
  return(strcpy(dane, tymczas));
}

void Tekst::pisz_tekst(void){
  cout << dane << endl;
}

int main(void) {
  Tekst tytul("\"Wygraj z C++\"");
  Tekst lekcja("Przeciazanie operatorow");

  tytul.pisz_tekst();
  tytul = tytul + " to najlepszy podrecznik.";
  tytul.pisz_tekst();

  lekcja.pisz_tekst();
  lekcja = lekcja - 'r';
  lekcja.pisz_tekst();
}
```

Jaki wynik uzyskałeś(-aś) po wykonaniu programu?

Dlaczego należało zmodyfikować funkcje operatorowe? Aby uzyskać wskazówkę do odpowiedzi na to pytanie zastąp instrukcje wersji 2 programu w liniach 45 i 49, odpowiednio instrukcjami wersji 3 z linii 45 i 49.

Pomogą Ci w tym poniższe rysunki: wersja 2 i wersja 3 – odpowiednio.

```
39
40 int main(void) {
41   Tekst tytul("\"Wygraj z C++\"");
42   Tekst lekcja("Przeciazanie operatorow");
43
44   tytul.pisz_tekst();
45   tytul + " to najlepszy podrecznik.";
46   tytul.pisz_tekst();
47
48   lekcja.pisz_tekst();
49   lekcja - 'r';
50   lekcja.pisz_tekst();
51 }
52
```

```
39
40 int main(void) {
41   Tekst tytul("\"Wygraj z C++\"");
42   Tekst lekcja("Przeciazanie operatorow");
43
44   tytul.pisz_tekst();
45   tytul = tytul + " to najlepszy podrecznik.";
46   tytul.pisz_tekst();
47
48   lekcja.pisz_tekst();
49   lekcja = lekcja - 'r';
50   lekcja.pisz_tekst();
51 }
52
```

Pozostałą część kodu tak zmodyfikowanego programu pozostaw bez zmian. Skompiluj zmodyfikowany program. Jakie komunikaty kompilatora otrzymałeś(-aś)?

Pamiętaj: gdy definiujesz własne operatory, to C++ pozostawia Ci swobodę wyboru sposobu używania/kodowania operatora. Jednak nie zapominaj, że celem przeciążania operatorów jest zwiększenie czytelności programów.

Zawracanie głowy…

Tekst programu operującego na zmiennych napisowych (tekstach) można w sposób krótki i prosty zapisać używając klasy **string**. Porównaj program poniżej.

```
//Rachunek napisow/tekstow - klasa string

#include <iostream>
using namespace std;

int main(void) {
  string zms, zm1, zm2;

  zm1="To poczatek napisu,";
  zm2=" a to jest koniec napisu.";

  zms = zm1 + zm2;

  cout << zm1 << endl;
  cout << zm2 << endl;
  cout << zms << endl;
}
```

Prosto i przejrzyście! Ale ktoś zakodował funkcje operatorowe klasy **string**! Poszukaj kodu źródłowego klasy **string**. Czy jego analiza jest lekka, łatwa i przyjemna?

Operatory związane są z daną klasą i dlatego możliwa jest **„wersja mixed”**:

TEKST KONTYNUOWANY NA STRONIE NASTĘPNEJ!

```
//przeciazanie operatorow (+) i (-) dla tekstow - mix

#include <iostream>
#include <string.h>
using namespace std;

class Tekst{
  public:
    Tekst(const char *);
    char *operator+(const char *);
    char *operator-(const char);
    void pisz_tekst(void);
  private:
                char dane[256];
};

Tekst::Tekst(const char *tks){
                strcpy(dane, tks);
}

char *Tekst::operator+(const char *tks){
                return(strcat(dane, tks));
}

char *Tekst::operator-(const char znak){
  char tymczas[256];
  int i, j;

  for(i=0, j=0; dane[i]; i++)
                if(dane[i] != znak)
                  tymczas[j++] = dane[i];
  tymczas[j] = '\0';
  return(strcpy(dane, tymczas));
}

void Tekst::pisz_tekst(void){
  cout << dane << endl;
}

int main(void) {
  Tekst tytul("\"Wygraj z C++\"");
  Tekst lekcja("Przeciazanie operatorow");
  string zms, zm1, zm2;

  zm1 = "To poczatek napisu,";
  zm2 = " a to jest koniec napisu.";

  tytul.pisz_tekst();
//Operator +
  tytul = tytul + " to najlepszy podrecznik.";
  tytul.pisz_tekst();

  lekcja.pisz_tekst();
  lekcja = lekcja - 'r';
  lekcja.pisz_tekst();

//Operator +
  zms = zm1 + zm2;

  cout << endl;
  cout << zm1 << endl;
  cout << zm2 << endl;
  cout << zms << endl;
}
```

Zwróć uwagę, że używamy operatora (**+**) do zmiennych różnych klas (kodując go tym samym znakiem graficznym), ale kompilator po typie zmiennej odwołuje się do właściwej **funkcji operatorowej**.

Zadanie:

Napisz program przeciążający **operator** (**==**) (**równe**) porównujący dwa teksty (zmienne lub stałe napisowe) i dający jako wynik wartość **1** gdy teksty są równe oraz **0** gdy porównywane teksty nie były równe.

Źródła:

1. Wykłady z MiTP, sem. 1, IT AGH 2021
2. Kris Jamsa, „Wygraj z C++, Wyd. Mikom, Warszawa 1996
3. Jerzy Grębosz, „Opus magnum C++11”, wyd. 2 popr., Wyd. Helion, Gliwice 2020
4. Bjarne Stroustrup, „Język C++. Kompendium wiedzy (C++11)”, wyd. 4, Wyd. Helion, Gliwice 2014