Podstawy programowania: Laboratorium nr 2

Pętle. Typy proste.

2017-2018

mgr inż. Przemysław Walkowiak
dr inż. Michał Ciesielczyk

Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie informacje ma podać oraz opisuj informację, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie).

Wprowadzenie

Konwersja typów

W języku C++ możemy zmienić typ dowolnej wartości poprzez zastosowanie odpowiedniej konwersji (oczywiście o ile konwersja jest możliwa), nazywanej również rzutowaniem (ang. *casting*). Należy również wiedzieć, że taka operacja może zmienić samą wartość. Na przykład podczas konwersji liczby zmiennoprzecinkowej na liczbę całkowitą zostanie obcięta część dziesiętna, tj. zamiast liczby 3.14 uzyskamy liczbę 3. Konwersja typów ma następującą składnię:

```
    static_cast < typ > ( x )
```

gdzie `x` jest zadeklarowaną zmienną, a `typ` typem na który rzutujemy.

Przykład:

```
float pojemnosc = 45.3;

cout << pojemnosc << endl;
// na ekranie zostanie wyswietlone "45.3"

cout << static_cast<int>(pojemnosc) << endl;
// na ekranie zostanie wyswietlone "45"
```

W programie może również nastąpić niejawna konwersja (czyli bez korzystania z rzutowania):

```
float pojemnosc = 45.3;
int pojemnosc_int = pojemnosc;

cout << pojemnosc_int << endl; // na ekranie zostanie wyswietlone "45"
```

Więcej informacji oraz przykładów można znaleźć tutaj: `http://en.cppreference.com/w/cpp/language/static_cast`.

Definiowanie stałych

Bardzo często w programista będzie chciał wykorzystać jakąś wartość liczbową/napisową bezpośrednio w kodzie. O ile napisy często mówią same za siebie co oznaczają, to w przypadku liczb już niekoniecznie. Aby nie męczyć czytającego kod próbą domyślania się co dana liczba oznacza, możemy ją po prostu nazwać.

Na przykład zamiast pisać takie wyrażenie:

```
float r;
std::cin >> r;
float pole_kola = 3.14159265359 * r * r;
float obwod_kola = 2 * 3.14159265359  * r;
```

można zdefiniować dodatkową stałą:

```
const float PI = 3.14159265359;

float r;
std::cin >> r;
float pole_kola = PI * r * r;
float obwod_kola = 2 * PI  * r;
```

Dodatkowo jeżeli będziemy chcieli zmienić dokładność wartości pi, wystarczy ją zmienić tylko w jednym miejscu. Modyfikator **const** zabezpiecza ponadto przed przypadkową modyfikacją.

Instrukcja warunkowa switch-case

Instrukcja warunkowa **switch-case** pozwala na warunkowe wykonanie innych instrukcji (podobnie jak instrukcja **if-else**). Składnia tej instrukcji jest następująca:

```
switch (expression) {
    case constant_expression: {
        statement1;
    }
    default: {
        statement2;
    }
}
```

gdzie:

- `expression` wyrażenie, do którego przyrównywane będą stałe z bloków **case**,
- `constant_expression` to stała liczbowa (np. **int**, **char**), do której przyrównana zostanie wartość `expression`,

- `statement1` to instrukcja (lub zestaw instrukcji), która zostanie wykonana, gdy wyrażanie `expression` będzie równe `constant_expression`,
- `statement2` to instrukcja (lub zestaw instrukcji), która zostanie wykonana domyślnie (tj. niezależnie od wartości `expression`).

Uwaga! W przeciwieństwie do instrukcji **if–else**, może się zdarzyć, że wykonane zostanie jednocześnie więcej niż jedna instrukcja. Aby temu zapobiec należy stosować instrukcję skoku **break**. Więcej informacji: `http://en.cppreference.com/w/cpp/language/switch`.

Pętle

W C++ pętle umożliwiają cykliczne wykonywanie ciągu instrukcji do momentu zajścia określonych warunków. Składnia pętli **for** wygląda następująco:

```
for (init_expression; cond_expression; loop_expression) {
    statement;
}
```

gdzie:

- `init_expression` to instrukcja wykonywana **na początku** pętli;
- `cond_expression` to warunek sprawdzany **przed** każdą iteracją pętli, jeśli nie zostanie spełniony pętla zostaje przerwana;
- `loop_expression` to instrukcja wykonywana **po** każdej iteracji pętli;
- `statement` to instrukcja(lub zestaw instrukcji), która będzie wykonywana z każdą iteracją pętli.

Składnia pętli **while** wygląda następująco:

```
while (cond_expression) {
    statement;
}
```

gdzie:

- `cond_expression` to warunek sprawdzany **przed** każdą iteracją pętli, jeśli nie zostanie spełniony pętla zostaje przerwana;
- `statement` to instrukcja(lub zestaw instrukcji), która będzie wykonywana z każdą iteracją pętli.

Składnia pętli **do–while** wygląda następująco:

```
do {
    statement
} while (cond_expression);
```

gdzie:

- `cond_expression` to warunek sprawdzany **po** każdej iteracji pętli, jeśli nie zostanie spełniony pętla zostaje przerwana;
- `statement` to instrukcja (lub zestaw instrukcji), która będzie wykonywana z każdą iteracją pętli.

Zadania

Zadanie 1

Wykorzystując pętlę **do–while** oraz konstrukcję **switch**, w funkcji głównej zaimplementuj menu wyboru zadania. Uruchamiany program powinien kolejno:

1. zapytać użytkownika które zadanie chce wykonać,
2. wywołać funkcję `zadanieX()`, gdzie X to numer zadania wybrany przez użytkownika; w przypadku wybrania nieodpowiedniego numeru zadania wyświetlić odpowiedni komunikat.
3. zapytać użytkownika czy chce zakończyć program,
4. wrócić odpowiednio do pkt 1. lub zakończyć działanie programu.

Możesz wykorzystać tę implementację na kolejnych laboratoriach.

Zadanie 2

Wczytaj od użytkownika dwie liczby, a następnie wykonaj na nich operacje dodawania, mnożenia, odejmowania i dzielenia. Wyświetl wynik każdej operacji na ekranie. Zadanie wykonaj w dwóch wariantach:

a) wykorzystując typy całkowite,
b) wykorzystując typy zmiennoprzecinkowe.

Sprawdź w wariancie pierwszym co się stanie, gdy przed dokonaniem operacji dzielenia zarzutuje się je na typ zmiennoprzecinkowy. Co się stanie gdy zarzutujesz tylko jedną z liczb? Wyświetl na ekranie odpowiedni komentarz.

Zadanie 3

Wypisz na ekranie rozmiar pamięci używany przez poszczególne proste typy danych: **bool**, **char**, **int**, **short**, **long**, **float**, oraz **double**. Sprawdź również czy i jaki wpływ na rozmiar zmiennej mają modyfikatory: **long**, **short**, **unsigned**, **signed**.

Wskazówka Aby wyświetlić na ekranie rozmiar pamięci używany przez dany typ lub wartość należy skorzystać z operatora **sizeof**. Przykładowo, aby wyświetlić rozmiar **int**: `cout << sizeof(int);`

Dodatkowe informacje:

- Podstawowe typy danych: `http://www.cplusplus.com/doc/tutorial/variables/#fundamental`.
- Operator **sizeof**: `http://en.cppreference.com/w/cpp/language/sizeof`

Zadanie 4

Wczytaj od użytkownika jedną liczbę zmiennoprzecinkową, a następnie wyświetl ją na ekranie:

a) z domyślną dokładnością (`std::defaultfloat`),
b) w notacji naukowej (`std::scientific`),
c) w systemie szesnastkowym (`std::hexfloat`),
d) z dokładnością do 3 miejsc po przecinku (`std::ios_base::precision`),
e) w notacji wykładniczej (naukowej), z dokładnością do 3 miejsc po przecinku.

Dodatkowe informacje:

- std::fixed, std::scientific, std::hexfloat, std::defaultfloat.
- std::ios_base::precision.

Zadanie 5

Korzystając z instrukcji warunkowych **switch**, napisz prosty kalkulator (działania $+$, $-$, $/$, $*$), który będzie wyświetlał użytkownikowi menu wyboru operacji, wczytywał odpowiednie argumenty oraz wypisywał wynik działania. Program ma kończyć działanie jedynie po wybraniu odpowiedniej pozycji z menu.

Wskazówka Aby wczytać pojedynczy znak z klawiatury możesz też skorzystać z `std::cin`:

```
char znak;
std::cin >> znak;
```

Zadanie 6

Wczytaj od użytkownika jedną liczbę całkowitą, a następnie wykonaj na niej operacje stosując operatory pre- i postinkremetacji oraz pre- i postdekrementacji. Zaobserwuj działanie wszystkich operatorów.

Wskazówka Aby zaobserwować różnicę pomiędzy operatorami pre- oraz post- umieść te operacje w linii wypisywania na ekranie, tj. `std::cout << ++i;`

Dodatkowe informacje:

- Operatory inkrementacji oraz dekrementacji.

Zadanie 7

Napisz program wypisujący na ekranie liczby całkowite od 1 do 100. Skorzystaj z:

a) pętli **for**,
b) pętli **while**,
c) pętli **do–while**,

Dodatkowe informacje:

- **for**, **while**, **do–while** - `http://cplusplus.com/doc/tutorial/control/`.

Zadanie 8

Wyświetl na ekranie wszystkie liczby podzielne przez 7 z zakresu $1 \ldots 500$.

Zadanie 9

Napisz program obliczający wyrażenie:

$$\sum_{i=1}^{10000} \frac{1}{i^2} \tag{1}$$

Sumowania dokonaj w kolejności od $i = 1$ do 10000 oraz od $i = 10000$ do 1. Porównaj wyniki. Czy są takie same?

Zadanie 10

Napisz program obliczający wyrażenie:

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots \tag{2}$$

Następnie pomnóż wynik przez 4 i wyświetl na ekranie. Jaka to liczba?

Wskazówka 1 Zamiast nieskończoności przyjmij bardzo dużą wartość.
Wskazówka 2 Do wyznaczania potęgi możesz wykorzystać funkcję `std::pow`.

Na następne zajęcia

- Funkcje – deklaracja, przekazywanie parametrów oraz zwracanie wyniku z/do funkcji: `http://en.cppreference.com/w/cpp/language/functions` oraz `http://www.cplusplus.com/doc/tutorial/functions/`
- Instrukcja **return**: `http://en.cppreference.com/w/cpp/language/return`
- Mechanizm rekurencji (aby zrozumieć rekurencję, trzeba wpierw zrozumieć rekurencję): `https://pl.wikipedia.org/wiki/Rekurencja`