Część XVIII C++ Funkcje

Widoczność zmiennych

Czy wartości każdej zmiennej można zmieniać w dowolnym miejscu kodu?
Czy można zadeklarować dwie zmienne o takich samych nazwach?

Umiemy już podzielić nasz program na mniejsze części - bloki instrukcji warunkowych, pętle. Potrafimy wyodrębnić z kodu często powtarzający się blok instrukcji i stworzyć z niego funkcję. W każdym z tych bloków instrukcji oraz w każdej funkcji prawie zawsze wykorzystuje się zmienne. Pojawia się więc problem zmiennych o tych samych nazwach, z których korzystają różne funkcje. Ten problem nie ominie żadnego programisty, więc warto poznać zasadę widoczności i dostępności zmiennych w programie.

Zmienne lokalne

Czasami wykorzystujemy daną zmienną tylko i wyłącznie wewnątrz określonego bloku instrukcji. A co, jeśli zmienna o tej samej nazwie była zdefiniowana przed tym blokiem. Nie martwy się. Zmienna zadeklarowana wewnątrz danego bloku instrukcji widoczna jest tylko w tym bloku. Spójrzmy na przykład

```
    for (int i = 0; i < 5; i++)
    {
        cout << i << endl;
    }

    cout << i << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

W pętli for została zadeklarowana zmienna **i** , która w każdym przebiegu pętli jest zwiększana o 1 i wyświetlana na ekranie. Pętla kończy się, jeśli **i** przyjmie wartość 5.

Jeśli poza pętlą chcemy sprawdzić, ile wynosi **i** , i wpiszemy instrukcję program nie zostanie skompilowany. Wynika to z tego, że zmienna **i** poza blokiem instrukcji pętli for nie jest znana (nie została zadeklarowana).

Ćwiczenie 1

1. Utwórz nowy projekt w Dev C++ i zapisz go na pulpicie w folderze o nazwie **nazwisko63**
2. Wprowadź do projektu modyfikacje tak aby wyglądał jak poniżej

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{

    for (int i = 0; i < 5; i++)
    {
        cout << i << endl;
    }

    cout << i << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

3. Z menu Uruchom wybieramy Kompiluj i uruchom.
4. Program nie zostanie skompilowany, **zamknij program i przejdź do dalszej części instrukcji.**

Przyjrzyjmy się innemu przykładowi

```
int main(int argc, char *argv[])
{
    int i = 5;
    cout << i << endl;

    {
        int i = 10;
        cout << i << endl;
    }

    cout << i << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Wewnątrz funkcji **main** deklarujemy zmienną o nazwie **i** i inicjujemy ją wartością 5.

Wyświetlamy wartość zmiennej **i** na ekranie.

Następnie wewnątrz bloku instrukcji ponownie definiujemy zmienną o nazwie **i** , przypisując jej wartość 10. Ponownie wyświetlamy wartość zmiennej **i** na ekranie.

Po zakończeniu bloku instrukcji jeszcze raz wyświetlamy wartość zmiennej |D ©. Po skompilowaniu i uruchomieniu programu na ekranie zobaczymy widok ^Jak więc widzimy, zmienna [D zadeklarowana (oraz zainicjowana wartością 10) wewnątrz bloku instrukcji przysłoniła zmienną |D (o wartości 5) zadeklarowaną poza tym blokiem i w żaden sposób nie wpłynęła na jej wartość (po zakończeniu bloku zmienna ponownie ma wartość 5).

Ćwiczenie nr 2

1. Utwórz nowy projekt w Dev C++ i zapisz go na pulpicie w folderze o nazwie **nazwisko64**
2. Wprowadź do projektu modyfikacje tak aby wyglądał jak poniżej

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    int i = 5;
    cout << i << endl;

    {
        int i = 10;
        cout << i << endl;
    }

    cout << i << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

3. Z menu Uruchom wybieramy Kompiluj i uruchom.

Umieszczając dowolne instrukcje pomiędzy nawiasami klamrowymi, tworzymy blok instrukcji.

Zmienne globalne

Zmienne zdefiniowane w bloku instrukcji widziane są tylko w obrębie tego bloku. Podobnie jest ze zmiennymi zdefiniowanymi wewnątrz funkcji - zmienna zadeklarowana na jej początku widoczna jest wyłącznie w tej funkcji. Ale co się stanie, jeśli naszą zmienną zdefiniujemy poza wszystkimi funkcjami, na przykład przed funkcją **main** . Sprawdźmy to.

Ćwiczenie nr 3

1. Utwórz nowy projekt w Dev C++ i zapisz go na pulpicie w folderze o nazwie **nazwisko65**
2. Wprowadź do projektu modyfikacje tak aby wyglądał jak poniżej

```
#include <cstdlib>
#include <iostream>

using namespace std;

int liczba = 10;

int test();

int main(int argc, char *argv[])
{
    cout << "Wartosc zmiennej liczba wewnatrz main: " << liczba << endl;

    liczba = 40;
    cout << "Nowa wartosc zmiennej liczba wewnatrz main: " << liczba << endl;

    int liczba2 = 0;
    liczba2 = test();
    cout << "Wartosc przypisana do zmiennej liczba2: " << liczba2 << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}

int test()
{
    cout << "Wartosc zmiennej liczba wewnatrz funkcji test: " << liczba << endl;
    return ++liczba;
}
```

Na początku wyświetlamy wartość zmiennej **liczba**, zmieniamy jej wartość i ponownie ją wyświetlamy.

Dzięki temu przekonamy się, czy zmienną **liczba** można wykorzystywać tak, jak zmienną zdefiniowaną wewnątrz funkcji **main** .

Po skompilowaniu okaże się, że ze zmiennej zadeklarowanej przed funkcją **main** możemy normalnie korzystać.

Wewnątrz funkcji **main** wywołujemy teraz bezargumentową funkcję **test()** której zadaniem będzie wyświetlenie wartości zmiennej **liczba** i zwrócenie tej wartości zwiększonej o 1. Jeśli taki kod zadziała, przekonamy się o tym, że zmienną **liczba** możemy wykorzystywać również wewnątrz innych funkcji. Jak się okaże po kompilacji i uruchomieniu, na ekranie zostanie wyświetlona prawidłowa wartość zmiennej **liczba.**

Na koniec wyświetlamy wartość zmiennej **liczba2** , do której przypisaliśmy wynik działania funkcji **test()** (która, jak pamiętamy, zwróciła zwiększoną o 1 wartość zmiennej **liczba**). Jak szybko zauważymy po skompilowaniu i uruchomieniu programu, ze zmiennej **liczba** można korzystać w każdej funkcji naszego programu - możemy ją bez problemów wyświetlać, zmieniać i przypisywać w każdym miejscu naszego kodu.

3. Z menu Uruchom wybieramy Kompiluj i uruchom.

Zmienną dostępną tylko w określonym fragmencie kodu (na przykład zadeklarowaną wewnątrz pojedynczego bloku instrukcji i tylko w nim dostępną) nazywamy zmienną lokalną.

Zmienne globalne zamiast argumentów

Skoro zmienne globalne widziane są we wszystkich funkcjach naszego programu, moglibyśmy nimi zastąpić argumenty przekazywane do tych funkcji. No bo po co przekazywać jakąś wartość do funkcji, skoro można uczynić z niej wartość globalną i wykorzystywać w każdej funkcji. Niestety, w taki sposób myśli wielu początkujących programistów, nagminnie wykorzystując zmienne globalne.

Kategorycznie powinniśmy się wystrzegać takich praktyk, ograniczając korzystanie ze zmiennych globalnych do minimum, czyli do sytuacji, w której zmienna globalna wykorzystywana jest przez większość funkcji w naszym kodzie, ale jej wartość nie jest przez nie zmieniana. Przykładem może być zmienna, której na początku działania programu przypisujemy wartość pobraną od użytkownika (na przykład jego imię) i później korzystamy z niej w różnych funkcjach naszego programu.

Jeśli zmienna została zadeklarowana poza funkcją main (oraz poza wszystkimi innymi

funkcjami) widoczna jest w całym naszym programie. Taką zmienną nazywamy zmienną globalną (w przeciwieństwie do zmiennych lokalnych widocznych tylko w określonych funkcjach czy blokach instrukcji).

Przydatne funkcje matematyczne

W programach komputerowych występuje zazwyczaj wiele obliczeń. Poznajmy kilka standardowych funkcji, które nam je ułatwią. Poznaliśmy już podstawowe operatory matematyczne. Za ich pomocą możemy wykonywać takie operacje, jak dodawanie, odejmowanie, mnożenie, dzielenie i dzielenie modulo. Niestety, w wielu wypadkach takie operacje nie będą nam wystarczały. Warto więc poznać kilka przydatnych funkcji matematycznych

Potęgowanie

Czasami zachodzi potrzeba podniesienia jednej liczby do określonej potęgi. Możemy do tego celu wykorzystać funkcję

```
float z;
float x = 2, y = 10;

z = pow(x, y);
```

Przy jej wywołaniu podajemy dwa argumenty - oba muszą być typu zmiennoprzecinkowego (jeśli przekazujemy do funkcji nie zmienną, lecz wartość liczbową, nawet jeśli jest całkowita, musimy postawić po niej kropkę i podać wartość ułamkową na przykład 0).

```
z = pow(2.0,10.0);
```

```
z = exp(3.0);
```

Przydatna może się okazać również funkcja , która podnosi często wykorzystywaną liczbę e (równą około 2,72) do ustalonej przez nas za pomocą argumentu potęgi (w tym wypadku argument również musi być typu zmiennoprzecinkowego). Obie funkcje zwracają wartość zmiennoprzecinkową.

Funkcje trygonometryczne

Sinus, cosinus czy tangens - bez możliwości obliczania wartości tych funkcji nie obędziemy się w wielu przypadkach (związanych na przykład z wyświetlaniem grafiki). Wystarczy wpisać skróconą nazwę funkcji trygonometrycznej i argument (typu zmiennoprzecinkowego).

```
z = sin(x);

z = cos(2.5);

z = tan(1.0);
```

Pierwiastek

Pierwiastek trudno obliczyć samodzielnie. Dlatego warto wiedzieć o istnieniu funkcji wyliczającej pierwiastek kwadratowy dowolnej liczby (musi mieć typ rzeczywisty).

```
z = sqrt(x);
```

Logarytmy

Umiemy już obliczać potęgi różnych liczb (w tym również liczby e). Warto jeszcze poznać funkcje logarytmujące. Za pomocą funkcji obliczymy logarytm naturalny podanej w argumencie liczby.

```
z = log(x);
```

Wykorzystując natomiast funkcję , obliczymy logarytm dziesiętny. W obu wypadkach argumenty muszą być typu zmiennoprzecinkowego.

```
z = log10(100.0);
```

Wszystkie pliki z nazwiskiem i kolejnym numerem umieszczamy w swoim folderze nazwiskocplusplus na serwerze.