Podstawy programowania w C++

Twój pierwszy program

Opracował: Andrzej Nowak

Bibliografia:

CPA: PROGRAMMING ESSENTIALS IN C++ https://www.netacad.com

Teraz napiszemy bardzo prosty (i jednocześnie całkowicie bezużyteczny) program napisany w języku C ++.
Program ten napiszemy w celu przedstawienia podstawowych zasad rządzących językiem.
Sam program będzie modyfikowany wiele razy, ponieważ wzbogaci się o dodatkowe elementy poszerzające naszą wiedzę programistyczną.

Najpierw musimy zdefiniować nasze oczekiwania dotyczące programu.
Chcemy, aby na ekranie pojawił się krótki tekst. Tekst będzie prosty:

To ja, twój pierwszy program.

Tego właśnie chcemy w tej chwili.
Jakie dalsze kroki musi wykonać nasz pierwszy program? Spróbujmy je wymienić tutaj:

1. musi się rozpocząć
2. napisać tekst na ekranie
3. musi się zatrzymać

Ten rodzaj ustrukturyzowanego i półformalnego opisu każdego etapu programu nazywa się algorytmem.

Czas zobaczyć nasz program. Oto jest →

```
#include <iostream>

using namespace std;

int main(void)
{
    cout<<"To ja twoj pierwszy program. ";
    return 0;
}
```

Wygląda to trochę tajemniczo, prawda? Sprawdźmy dokładnie każdą linię programu i odkryjmy jego znaczenie i cel. Opis nie jest szczególnie dokładny, a ci, którzy znają już trochę C ++, prawdopodobnie dojdą do wniosku, że jest zbyt uproszczony i nieco dziecinny.
Zrobiliśmy to celowo - nie budujemy Rzymu w jeden dzień. Nawet za tydzień!

Zwróć uwagę na znak # (hash) na początku pierwszego wiersza. Oznacza to, że treścią tej linii jest tzw. **dyrektywa preprocesora** - jest to oddzielna część kompilatora, której zadaniem jest wstępne odczytanie tekstu programu i wprowadzenie w nim pewnych modyfikacji.

Prefiks "pre" sugeruje, że te operacje są wykonywane przed pełnym przetwarzaniem (kompilacją).

Zmiany wprowadzone przez preprocesor są w pełni kontrolowane przez jego dyrektywy.

```
#include <iostream>
```

W naszym programie mamy do czynienia z dyrektywą włączającą. Kiedy preprocesor spełnia tę dyrektywę, zastępuje dyrektywę zawartością pliku, którego nazwa znajduje się w dyrektywie (w naszym przypadku jest to plik o nazwie `iostream`).

Zmiany dokonane przez preprocesor nigdy w żaden sposób nie modyfikują zawartości pliku źródłowego. Wszelkie zmiany wprowadzane są na niestabilnej kopii programu, która znika natychmiast po zakończeniu pracy kompilatora.

Dlaczego preprocesor musi zawierać zawartość zupełnie nieznanego pliku o nazwie iostream?

Pisanie programu jest podobne do budowy konstrukcji z gotowych bloków.

W naszym programie użyjemy jednego takiego bloku do napisania czegoś na ekranie. Ten blok nazywa się `cout` (możesz go znaleźć w naszym kodzie), ale kompilator nic o nim nie wie.

W szczególności kompilator nie ma pojęcia, że `cout` jest prawidłową nazwą dla tego bloku, do puki nie jest to oznaczone. Kompilator musi być ostrzeżony o tym - musi być tego świadomy.

Zestaw wstępnych informacji, których potrzebuje kompilator, jest zawarty w plikach nagłówkowych. Pliki te zawierają zbiór wstępnych informacji o gotowych blokach, które mogą być używane przez program do pisania tekstu na ekranie lub do czytania liter z klawiatury. Kiedy więc nasz program zamierza coś napisać, użyje bloku o nazwie `cout`, który jest w stanie wyświetlić napis. Nie chcemy, aby kompilator był zaskoczony, więc musimy go ostrzec. Twórcy kompilatora umieścili zestaw tych przewidywalnych informacji w pliku `iostream`. Wszystko, co musimy zrobić, to użyć pliku. Dokładnie tego oczekujemy od dyrektywy `include`.

Ale gdzie znajduje się plik `iostream`?

Preprocesor wie, gdzie to jest.

W języku C ++ wszystkie elementy standardowej biblioteki C ++ są zadeklarowane w przestrzeni nazw o nazwie **`std.`**

Przestrzeń nazw jest abstrakcyjnym kontenerem lub środowiskiem stworzonym do przechowywania logicznego grupowania unikatowych elementów (bloków).

Jednostka zdefiniowana w przestrzeni nazw jest powiązana tylko z tą przestrzenią nazw.

Jeśli chcesz używać wielu standardowych elementów C ++ musisz wstawić instrukcję `using namespace` u góry każdego pliku, poza jakąkolwiek funkcją.

```
using namespace std;
```

Instrukcja powinna określać nazwę pożądanej przestrzeni nazw (w naszym przypadku **`std)`**. W ten sposób standardowe wyposażenie będzie dostępne w całym programie.

Powiedzieliśmy już coś o blokach. Teraz chodźmy trochę głębiej.

Jednym z najczęstszych typów bloków używanych do budowania programów w C ++ są funkcje.

Jeśli kojarzysz funkcję z matematyką, jesteś na dobrej drodze.

Wyobraź sobie funkcję jako czarną skrzynkę, do której można coś włożyć (choć nie zawsze jest to konieczne) i wyjąć z niej coś nowego, jak z magicznego kapelusza.

- Rzeczy do umieszczenia w skrzynce nazywają się argumentami funkcji (lub parametrami funkcji).
- Rzeczy, które należy wyjąć z pudełka, nazywane są wynikami funkcji.

Standard języka C ++ zakłada, że pośród wielu różnych bloków, które można umieścić w programie, zawsze musi być obecny jeden określony blok, w przeciwnym razie program nie będzie poprawny. Ten blok jest zawsze funkcją o tej samej nazwie: **`main`**.

```
int main(void)
{

}
```

Każda funkcja w C ++ zaczyna się od następującego zestawu informacji:

- Jaki jest rezultat funkcji?
- Jak nazywa się funkcja?
- Ile parametrów ma funkcja i jakie są ich nazwy?

Przyjrzyj się uważnie naszemu programowi i postaraj się go dokładnie przeczytać, akceptując fakt, że nie zrozumiesz jeszcze wszystkiego.

- wynik funkcji jest wartością całkowitą (możemy odczytać ją ze słowa int, które jest skrótem dla liczby całkowitej)
- nazwa funkcji to - główna (**`main`**)
- funkcja nie wymaga żadnych parametrów (informuje o tym słowo `void`).

Ten zestaw informacji jest czasami nazywany prototypem i jest jak etykieta umieszczona w funkcji informującej o tym, jak możesz użyć tej funkcji w swoim programie.

Prototyp nie mówi nic o tym, co funkcja ma zrobić. Jest zapisany wewnątrz funkcji.

Wnętrze funkcji nazywa się ciałem funkcji.

Funkcja zaczyna się pierwszym nawiasem otwierającym **{** i kończy się odpowiednim nawiasem zamykającym **}** .

Treść funkcji może być pusta, co oznacza, że funkcja nie wykonuje dokładnie nic.

Możemy nawet utworzyć funkcję, która jest leniwą - może być zakodowana w ten sposób:

```
void leniwa (void)
{
}
```

Ta funkcja nie robi dokładnie nic – stąd nazwa `leniwa`

Wewnątrz głównego korpusu funkcji musimy napisać, co ma robić nasza funkcja (a więc i program). Jeśli zajrzymy do środka, znajdziemy odniesienie do bloku o nazwie cout.

Każda instrukcja (a dokładniej każda instrukcja) w C ++ musi kończyć się średnikiem - bez niej program będzie niepoprawny.