Podstawy programowania w C++

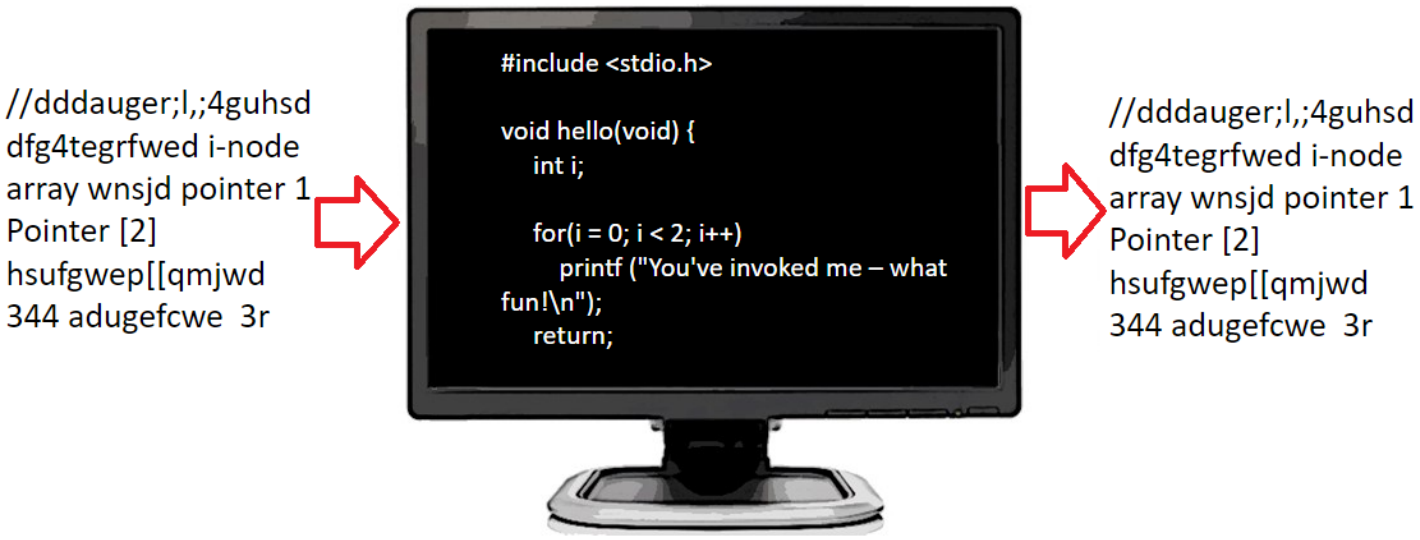
Strumienie wejścia – cin>> i wyjścia – cout<<

Opracował: Andrzej Nowak

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

Gdy dane są wprowadzane do programu przez użytkownika (człowieka) lub wczytywane z plików, nazywa się to **wejściem**.

Dane przesyłane w przeciwnym kierunku, tj. z komputera na ekran lub do pliku, nazywa się to **wyjściem.**



Poznaliśmy już jeden ze strumieni do wyprowadzania danych na ekran (**`cout`**) i używaliśmy go wraz z operatorem **<<**.

Operator **<<** jest czasami nazywany **operatorem wstawiania**, gdyż wstawia ciąg znaków do urządzenia znakowego (na przykład konsoli).

Zarówno operator << jak i strumień `cout` są odpowiedzialni za dwie ważne akcje:

- przekształcenie wewnętrznej (maszynowej) reprezentacji wartości całkowitej w formę akceptowalną dla ludzi
- przeniesienie skonwertowanej postaci na urządzenie wyjściowe, np. konsola

Strumienie są bardzo wydajnymi i wygodnymi narzędziami zarówno do wejścia, jak i wyjścia. Mogą łatwo wyprowadzać różne wartości różnych typów i mieszać je z tekstem. Mogą również łatwo wprowadzić wiele wartości naraz.

Przykład:

Wyświetl na ekranie wartość zmiennej **int**.

```
int zmienna = 110;
cout << zmienna;
```

Możesz także połączyć więcej niż jeden operator << w jednym ciągu `cout`, a każdy z drukowanych elementów może być innego rodzaju i mieć inny charakter.

```
int zmienna = 110;
cout <<"Wartosc "<< zmienna;
```

cout może wypisywać nie tylko napisy i wartości zmiennych, ale również wyniki obliczeń.

Spójrz na przykład poniżej.

```
int rozmiar_zmiennej = 123;
cout << "Elementy policzone do tej pory "
     << rozmiar_zmiennej * 4;
```

Manipulatory

Jeśli chcesz, aby wartość typu **int** była prezentowana jako stała liczba szesnastkowa, musisz użyć tak zwanego manipulatora.

Manipulator jest specjalnym rodzajem podmiotu, który informuje strumień, że formularz danych musi zostać natychmiast zmieniony. Wszystkie elementy wyprowadzone po aktywacji manipulatora zostaną przedstawione w żądanej formie.

Manipulator zaprojektowany do przełączania strumienia w tryb szesnastkowy nazywany jest **hexem**.

Przykład:

```
int byte = 255;

cout << ”Bajt w zapisie szesnastkowym ” << byte << hex;
```

Fragment wyświetli ciąg znaków składający się ze znaków "F" i "F".

Technicznie, manipulator jest funkcją, która zmienia jedną z właściwości strumienia wyjściowego, zwaną **basefield**.

Właściwość służy do określenia, która liczba powinna być użyta jako baza podczas konwersji wartości **int** do tekstu czytelnego dla człowieka.

UWAGA:

1. każdy manipulator rozpoczyna pracę od miejsca, w którym został umieszczony i kontynuuje swoją pracę nawet po zakończeniu instrukcji **cout**; kończy działanie tylko wtedy, gdy inny manipulator anuluje jego działanie;
2. nazwa manipulatora może być w konflikcie z dowolną inną nazwą zadeklarowaną przez programistę;
    - na przykład możesz mieć własną zmienną o nazwie **hex**, która może ukryć nazwę manipulatora;
    - takie konflikty są rozwiązywane przez wyspecjalizowany mechanizm nazywany przestrzenią nazw.

Manipulatory – zakres działania

dec

Manipulator **dec** przełącza strumień do postaci dziesiętnej. W większości przypadków nie jest to deklarowane jawnie, ponieważ wartość dziesiętna jest domyślnym trybem roboczym dla strumieni wyjściowych.

```
int bajt = 255;
cout << hex <<bajt;
cout << bajt << dec << bajt;
```

Przedstawiony przykład pokazuje, jak manipulatorzy zaczynają i kończą swoją pracę.

Podany fragment wyprowadzi trzy przykłady o tej samej wartości (sprawdź jakie)

`FF` jako szesnastkowa reprezentacja `255` (jako efekt manipulatora szesnastkowego)

`FF` ponownie (poprzednia aktywacja `hex` nadal działa)

`255` (w wyniku aktywacji dekodera `dec`)

oct

Manipulator **oct** przełącza strumień na tryb ósemkowy.

```
int bajt = 255;
cout << dec << bajt;
```

W przedstawiony przykładzie zostanie wyświetlona wartość 377 na ekranie, ponieważ $255_{(10)}$ to $377_{(8)}$

setbase

Trzy manipulatory, które wcześniej poznaliśmy, są tylko jedną z metod (prawdopodobnie najprostszą) dostępu do właściwości `basefield`.

Ten sam efekt można osiągnąć za pomocą manipulatora **`setbase`**, który bezpośrednio instruuje strumień o wartości bazowej, której powinien użyć podczas konwersji.

Jedynymi akceptowalnymi wartościami dla parametru `setbase` są 8, 10 i 16.

Przedstawiony fragment programu pokazuje użycie manipulatora `setbase`.

Uwaga:

Aby używać manipulatora `setbase` należy zadeklarować plik nagłówkowy o nazwie `iomanip`

```
#include <iostream>
#include <iomanip>

using namespace std;

int main(void)
{
int bajt = 255;
cout << setbase(16) <<bajt;
return 0;
}
```

Strumień wyjścia – cout – rozpoznawanie typów zmiennych

Strumienie wyjściowe (w tym `cout`) są w stanie rozpoznać rodzaj drukowanej wartości i odpowiednio działać, tj. wykorzystają odpowiednią formę prezentacji danych dla wartości `char` i `float`.

```
char znak = 'X', minus = '-';
float liczba_rzeczywista = 2.5;
cout << znak <<minus << liczba_rzeczywista;
```

wynik wykonania fragmentu programu:

```
X-2.5
```

`cout` jest w stanie rozpoznać rzeczywisty typ swojego elementu, nawet jeśli jest to efekt konwersji.

Oznacza to, że możemy zobaczyć kod ASCII dowolnego znaku zapisanego w zmiennej `char` i odwrotnie, lub zobaczyć znak, którego kod ASCII jest umieszczony wewnątrz zmiennej `int`.

```
char znak = 'X';
int wartosc = znak;
cout << znak <<"  " <<(int)znak<<" "<<wartosc<<" "<<(char)wartosc;
```

wynik wykonania fragmentu programu:

```
X 88 88 X
```

Czasami możemy chcieć (a czasami musimy) złamać linię przesyłaną na ekran.

Kiedy przedstawiamy wiele różnych wyników jeden po drugim w tym samym wierszu tekstu, jest to nieczytelne. Jedna linia jest w porządku, ale tysiąc wierszy napisanych w ten sposób sprawi, że oślepniesz.

Możemy przerwać linię na dwa sposoby.

Po pierwsze, możemy użyć jednego ze znaków kontrolnych zwanych "*newline*" i zakodowanych jako `\n`.

Znak nowej linii zmusza konsolę do ukończenia bieżącej linii i rozpoczęcia nowej.

Możemy osiągnąć dokładnie taki sam efekt, używając manipulatora o nazwie **`endl`** (jako "linia końcowa").

Strumień wejścia – cin – rozpoznawanie typów zmiennych

Równie ważne jak dane wyjściowe jest wprowadzanie danych. Właściwie trudno wyobrazić sobie nietrywialny program, który nie wymaga żadnych danych od użytkownika, chociaż można wykonać następujące czynności:

- zakodować wszystkie dane potrzebne w kodzie źródłowym (co czasami jest nazywane twardym kodowaniem)
- kiedy trzeba powtórzyć wykonanie programu z innymi danymi, wystarczy zmodyfikować program, skompilować go i uruchomić ponownie.

To nie jest szczególnie wygodne rozwiązanie. Znacznie lepiej jest uzyskać informacje od użytkownika, przenieść je do programu, a następnie użyć do obliczeń.

W jaki więc sposób program języka C ++ pobiera dane od człowieka i zapisuje je w zmiennych?

Najprostszym sposobem jest mentalna zmiana kierunku transferu i potwierdzenie, że dla wprowadzania danych:

- używamy strumienia **`cin`** zamiast **`cout`**
- używamy operatora **>>** zamiast **<<**.

Operator **>>** jest często nazywany operatorem ekstrakcji.

Strumień **`cin`**, wraz z operatorem ekstrakcji, jest odpowiedzialny za:

- przenoszenie czytelnej dla człowieka postaci danych z urządzenia wejściowego, np. konsola
- przekształcenie danych w wewnętrzną (maszynową) reprezentację wprowadzanej wartości.

Przykłady użycia strumieni `cin` i `cout` oraz operatorów >>,<<;

```
//program oblicza kwadrat danej liczby
#include <iostream>
using namespace std;

int main(void)
{
	int wartosc,kwadrat;

	cout << " Podaj jakas liczbe calkowita - oblicze jej kwadrat";
	cin >> wartosc;
	kwadrat = wartosc * wartosc;
	cout << " Podales liczbe " << warotsc << endl;
	cout << " Kwadrat tej liczby wynosi" << kwadrat << endl;
	return 0;
}

//program oblicza pierwiastek kwadratowy danej liczby
#include <iostream>
#include <cmath>
using namespace std;

int main(void) {
float wartosc,pierwiastek;

cout << " Podaj jakas liczbe - oblicze jej pierwiastek kwadratowy";<< endl;
cin >> wartosc;
if(wartosc >= 0.0) {
	pierwiastek = sqrtf(wartosc);
	cout << " Podales liczbe: " << wartosc << endl;
	cout << "Pierwiastek kwadratowy tej liczby wynosi: " << pierwiastek << endl;
	}
	return 0;}
```