**Dev-C++ - krótka instrukcja obsługi**

Po uruchomieniu kompilatora tworzymy nowy plik za pomocą wyboru z menu rozwijanego **Plik**, najpierw **Nowy**, a następnie **Plik źródłowy.**

Teraz wpisujesz treść pliku. Teraz pozostaje zapisać plik, wybierając z menu rozwijanego **Plik** opcję **Zapisz jako**. Teraz wpisujesz swoją nazwę pliku.

Zauważ, że pod miejscem na wpisywanie nazwy, jest wybrana opcja **CPP source (\*.cpp;\*.cc;\*.cxx;\*.c++;\*.cp)**. Ponieważ chcemy skompilować właśnie plik źródłowy, więc ta opcja jest prawidłowa. Jeśli do nazwy pliku nie dodamy jawnie rozszerzenia, to kompilator zapisze nasz plik z rozszerzeniem **cpp**. W zasadzie możemy nie podawać nazwy rozszerzenia pliku, bowiem w windowsie właśnie to rozszerzenie jest najczęściej uznawane za rozszerzenie pliku źródłowego w C++.

Po zapisaniu pliku, pozostaje spróbować go skompilować. Kompilację możemy przeprowadzić na różne sposoby: możemy wybrać z górnego menu rozwijanego **Uruchom** opcję **Kompiluj**, możemy też kliknąć odpowiednią ikonę na pasku narzędziowym lub możemy skorzystać ze skrótu klawiaturowego **CTRL + F9**.

Jeśli kompilacja się powiedzie, pojawi się okienko, na którym pisze m.in. **Status: done**. Klikamy zamknij - nasz program jest skompilowany. Jeśli natomiast są jakieś błędy (lub ostrzeżenia), wówczas wspomniane okienko tylko mignie przed naszymi oczami. Wysunie się natomiast dolne menu, w którym przeczytamy o błędach (ostrzeżenia są poprzedzone ciągiem **[Warning]**). W przypadku błędów poprawiamy kod programu i ponownie kompilujemy. W przypadku ostrzeżeń nie musimy kompilować - program został skompilowany.

Aby uruchomić program, możemy wybrać z górnego paska **Uruchom** pozycję **Uruchom**. Możemy też skorzystać z ikony lub ze skrótu **CTRL + F10**. Innym sposobem uruchomienia programu jest znalezienie go na dysku i podwójne kliknięcie w odpowiednią ikonę.

Ostatni sposób jest najmniej wygodny, za to najbardziej zalecany. Na pasku narzędziowym systemu windows klikamy **START**, a następnie **Uruchom**. Wpisujemy **cmd** lub **command** (dla starszych wersji windows). Następnie korzystając z dwóch poleceń **cd** i **dir**, przechodzimy do katalogu, w którym znajduje się skompilowana wersja programu. Program uruchamiamy wpisując jego nazwę (która jest taka sama jak nazwa programu źródłowego, tylko ma rozszerzenie **exe**). Ta metoda jest najbardziej zalecana, bowiem uzyskujemy dokładnie identyczny kod źródłowy dla systemu windows i linux. Jeśli uruchamiamy program pozostałymi metodami w windows, zazwyczaj tylko mignie nam przed oczami czarne okienko z programem - żeby temu zaradzić, musimy zawsze przed kończącym nawiasem klamrowym w kodzie źródłowym dodać raz (lub większą liczbę razy jeśli to nie pomaga) komendę **getchar();** i oczywiście ponownie skompilować program

**Pierwszy program**

Poniżej znajduje się pierwszy program w C++ jaki napiszesz:

**#include<iostream>**

**#include <string>**

**using namespace std;**

**int main ()**

**{**

**string imie;**

**cout <<"Podaj imie: ";**

**cin >>imie;**

**cin.ignore();**

**cout <<"Witaj "<<imie<<'\n';**

**cout <<"Gratulacje. To Twoj pierwszy program!"<<'\n';**

**cout <<"Nacisnij ENTER aby zakonczyc"<<'\n';**

**getchar();**

**return 0;**

**}**

Przepisz teraz powyższy kod programu, skompiluj go i uruchom. Jeśli w momencie kompilacji kompilator informuje Cię, że są jakieś błędy, to upewnij się, że wszystkie średniki są na swoim miejscu

Jeśli juz uruchomisz program, podaj swoje imię i naciśnij ENTER. Program wypisze kilka komunikatów oraz będzie czekał znowu, aż naciśniesz ENTER. Wówczas program zakończy swoje działanie.

**Ogólna budowa programu w C++**

Mimo, że programy napisane w języku C++ mogą być bardzo różnej długości, wszystkie jednak mają

następującą budowę:

• **dyrektywy preprocesora**

Wszystkie dyrektywy preprocesora poprzedzone znakiem #. Wszystkie dyrektywy preprocesora są wykonywane przez specjalny program zwany preprocesorem stąd ich nazwa. Na szczęście w większości kompilatorów (o ile nie we wszystkich) nie musimy się martwić tym, że dyrektywy te wykonuje preprocesor. Kompilator automatycznie uruchamia preprocesor gdy wykryje dyrektywę. Najczęściej wykorzystywaną dyrektywą preprocesora jest dyrektywa include - służy ona do dołączania plików nagłówkowych. Inną popularną dyrektywą jest define - służy ona do tworzenia nazw symbolicznych.

Aby dołączyć jeden ze standardowych plików języka C++ należy napisać:

**#include <iostream>**

• **wykorzystywane przestrzenie nazw**

W nieco bardziej skomplikowanych programach, aby ułatwić sobie życie, grupuje się pewne elementy w przestrzenie nazw. Również wszystkie elementy zdefiniowane standardowo w języku C++ są zdefiniowane w przestrzeni nazw - a konkretnie przestrzeni nazw o nazwie std. Aby włączyć dowolną przestrzeń nazw wystarczy napisać:

**using namespace nazwa;**

gdzie nazwa jest nazwą wybranej przestrzeni nazw.

• **funkcje zdefiniowane przez u**ż**ytkownika**

Ponieważ programy pisane w języku C++ mogą być bardzo duże, dlatego też wymyślono mechanizm, aby grupować pewne fragmenty kodu w funkcje. Dzięki temu znacznie łatwiej poruszać się po takim programie czy też pisać programy w kilka osób. Funkcjami zajmiemy się na nieco dalszym etapie nauki.

• **główna funkcja programu**

Mimo, że funkcja ta działa jak zwykła funkcja ma ona dwie bardzo charakterystyczne cechy. Po pierwsze funkcja ta musi mieć zawsze nazwę main - tak ktoś sobie wymyślił, że główna funkcja programu tak się nazywa i tak już pozostało do dzisiaj. Drugą bardzo charakterystyczną cechą jest to, że od tej funkcji zaczyna się całe wykonanie programu. Najprostsza funkcja main (która nic nie robi), wygląda tak:

**int main ()**

**{**

**return 0;**

**}**

Jeśli chcemy, aby nasz program cokolwiek wykonał, musimy umieścić dodatkowy kod pomiędzy **{** oraz **return 0;**

**Dyrektywa include**

Dyrektywa include jest jedną z najczęściej wykorzystywanych dyrektyw preprocesora. Za jej pomocą dołączamy do programu wybrany plik nagłówkowy.

Zastanawiasz się pewnie, po co włączać do programu jakieś pliki nagłówkowe. Otóż w plikach nagłówkowych zdefiniowane są elementy, z których prawie na pewno będziesz chcieć korzystać. Bez kilku czy kilkunastu pliku nagłówkowych napisanie nawet prostego programu byłoby w zasadzie niemożliwe. Niech przykładem będzie to, że aby nawet coś wypisać na ekran (tak jak to miało miejsce w naszym pierwszym programie), należy dołączyć właściwy plik nagłówkowy.

Niestety w początkowym etapie nauki dużym problemem jest jaki plik należy włączyć, aby można było coś zrobić. Otóż co gdzie się znajduje uda Ci się nauczyć, gdy będziesz często korzystać z pewnych elementów. Przydatne mogą być też pliki pomocy dołączone do Twojego kompilatora. Dyrektywa include występuje w dwóch postaciach:

**#include <nazwa>**

oraz

**#include "nazwa"**

W pierwszym wypadku dołączamy plik nagłówkowy, który znajduje się w miejscu określonym przez kompilator. Z takiego dołączania będziemy najczęściej korzystać. W drugim wypadku dołączamy plik znajdujący się w bieżącym katalogu - ta możliwość jest wykorzystywana na nieco wyższym poziomie nauki.

W obu przypadkach nazwa jest nazwą pliku, który chcemy dołączyć do programu. W tym miejscu następuje jednak małe zamieszanie. Wszystkie pliki nagłówkowe mają standardowo rozszerzenie h. Otóż pliki nagłówkowe pochodzące z czystego języka C++ dołączamy podając samą nazwę pliku bez rozszerzenia. Natomiast pliki pochodzące tak naprawdę z poprzednika języka C++, czyli języka C włączamy podając nazwę wraz z rozszerzeniem.

Dla przykładu:

**#include <iostream>**

ale

**#include <stdio.h>**

Mimo, że na razie z tego i tak nie będziesz korzystać, to warto dodać, że wykorzystując drugą

metodę dołączania plików nagłówkowych, z wykorzystaniem **"**, musimy dodać rozszerzenie pliku (niekoniecznie h) oraz, że możemy dołączać pliki znajdujące się nie tylko w bieżącym katalogu. Na przykład:

**#include "wazne/moj.hpp"**

spowoduje dołączenie pliku o nazwie moj.hpp znajdującego się w katalogu wazne, przy czym katalog wazne znajduje się w bieżącym katalogu.

**Wykorzystywane przestrzenie nazw**

Przestrzenie nazw ułatwiają grupowanie pewnych elementów. Wszystkie standardowe elementy języka są umieszczone w przestrzeni nazw std Dlatego też warto zadeklarować, że chcemy wykorzystywać tę przestrzeń nazw:

**using namespace std;**

Mimo, że możemy wykorzystać tą metodę, wcale nie musimy. Jednak deklarując wykorzystywanie jakiejś przestrzeni nazw, ułatwiamy sobie znacznie zapis w dalszej części programu. Dlatego mimo pewnych niekorzystnych skutków takiego działania uważam, że warto zawsze dołączać na początku programu standardową przestrzeń nazw std.

Można też zrezygnować z deklaracji użycia przestrzeni nazw. Wtedy jednak w programie, który przedstawiłem Ci w pierwszej lekcji, należałoby wszędzie zamiast cout napisać std::cout oraz wszędzie zamiast cin napisać std::cin.

UWAGA: Jeśli zapomnisz włączyć standardową przestrzeń nazw std, nie zdziw się, jeśli kompilator będzie wypisywać bardzo dziwne błędy. Czasami zorientowanie się, skąd się wziął błąd może zająć znacznie więcej czasu niż pisanie samego programu. Dlatego radzę Ci dobrze - nie zapominaj o dołączaniu przestrzeni nazw std!

**Główna funkcja programu**

Mimo, że w programie możemy mieć wiele różnych funkcji, zawsze musimy umieścić główną funkcję o nazwie main. Jest to jedyna funkcja, która jest uruchamiana automatycznie. Wszystkie pozostałe funkcje musimy uruchomić sami.

Wiesz już jak wygląda funkcja main i wiesz, że aby program coś wykonywał, to coś musi się znajdować właśnie w funkcji main. Na razie nie będę tłumaczył do czego służą tajemnicze int oraz (). W nawiasach klamrowych umieszczamy właściwą treść funkcji, czyli treść naszego programu.

Warto zwrócić uwagę na tajemniczą linię:

**return 0;**

Otóż ta tajemnicza linia, umieszczona przed nawiasem kończącym treść funkcji, czyli przed } zwraca wartość wywołania naszego całego programu. Wartość taka jest wykorzystywana przez system operacyjny do stwierdzenia, czy program zadziałał poprawnie, a jeśli nie, to co spowodowało błąd.

Przyjęło się, że poprawne działanie programu jest sygnalizowane zwróceniem wartości 0, dlatego też taką wartość będziemy zwracać. Ponieważ w tym kursie będziemy pisać proste programy, dlatego też możesz przyjąć, że w funkcji main należy zawsze zwracać wartość 0. Mimo, że niektóre kompilatory nie sygnalizują błędu, jeśli ta linia zostanie pominięta, jest to linia dosyć ważna i lepiej o niej nie zapominać, zwłaszcza jeśli byśmy chcieli pisać programy, które ktoś w przyszłości będzie wykorzystywać.

**Najprostszy program w C++**

**int main ()**

**{**

**return 0;**

**}**

Powyższy program to najprostszy program. Składa się on tylko z funkcji main, wewnątrz której nic nie robimy, czyli sam program faktycznie też nic nie robi. Ponieważ program nic nie robi, nie musimy dołączać ani żadnego pliku nagłówkowego ani standardowej przestrzeni nazw std.

**Schemat prostego programu**

**#include<iostream>**

**using namespace std;**

**int main ()**

**{**

**return 0;**

**}**

To jest schemat kodu programu, jaki my zazwyczaj będziemy wykorzystywać. Oczywiście w funkcji main będziemy umieszczać dodatkowe instrukcje.

**Schemat złożonego programu**

**#include<iostream>**

**#include <string>**

**using namespace std;**

**int main ()**

**{**

**return 0;**

**}**

Tak naprawdę ten schemat wcale nie jest złożony. Różni się on jedynie od poprzedniego tym, że oprócz pliku nagłówkowego iostream, dołączamy dodatkowo plik nagłówkowy string. Czasami bowiem będzie zdarzać się, że będziemy musieli dołączyć dwa (tak jak tutaj) lub większą ilość plików nagłówkowych, aby w sposób wygodny i szybki napisać program.

**Komentarze w programach**

Oprócz przedstawionych elementów każdego programu, można wyróżnić jeszcze jeden element - komentarze. W przeciwieństwie jednak do wymienionych wcześniej elementów, komentarze nie mają żadnego znaczenia dla kompilatora.

Jedynym powodem stosowania komentarzy jest wygoda osoby piszącej programy. Co prawda pisząc program, trzeba będzie poświęcić dodatkowy czas na napisanie komentarza, jednak pomyśl jak dużo czasu zyskasz, gdy za kilka dni, tygodni czy miesięcy znów spojrzysz na program i uda Ci się bez problemu zrozumieć zawiłe fragmenty kodu właśnie dzięki komentarzom.

Stosowanie komentarzy ponadto jest niezbędne również, gdy nad programem pracuje kilka osób. Także pisząc programy, których fragmenty będziesz wykorzystywać w przyszłości w innych programach, komentarze są niezbędne.

Komentarze można zapisywać w programach na dwa sposoby:

**/\*To jest**

**komentarz kilkulinijkowy**

**\*/**

lub

**// To jest komentarz jednolinijkowy.**

Pierwsza metoda służy do dłuższych komentarzy lub takich komentarzy, które chcemy, aby zajmowały tylko pewną część linii. Cały komentarz mieści się między **/\*** oraz **\*/**. Znaki rozpoczynające komentarz mogą być rozmieszczone dowolnie - mogą być w tej samej linii co tekst komentarza, mogą być w różnych liniach lub mogą się znajdować w jednej linii.

Druga metoda służy do komentarzy jednolinijkowych. Komentarz rozciąga się od **//** aż do końca linii. Nie można komentarza zakończyć przed końcem linii. Dlatego jeśli chcemy zakończyć komentarz wcześniej, a za nim umieścić normalny kod programu, musimy wówczas użyć pierwszej metody.

Komentarze mogą być ponadto również bardzo przydatne podczas testowania programu. Jeśli nie wiemy gdzie jest błąd w naszym programie, możemy pewne części programu otoczyć komentarzem - wówczas ta część programu się nie wykona. Otaczając coraz więcej kodu komentarzami, uda nam się w końcu zmniejszyć kod programu tak bardzo, że będziemy mogli stwierdzić, że tam właśnie jest błąd. Po poprawieniu błędu należy wtedy oczywiście usunąć wszystkie znaki komentarzy, tak aby wykonywał się cały kod programu.

**Jak działa Twój pierwszy program**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main ()**

**{**

**string imie;**

**cout <<"Podaj imie: ";**

**cin >>imie;**

**cin.ignore();**

**cout <<"Witaj "<<imie<<'\n';**

**cout <<"Gratulacje. To Twoj pierwszy program!"<<'\n';**

**cout <<"Nacisnij ENTER aby zakonczyc"<<'\n';**

**getchar();**

**return 0;**

**}**

Na początku dołączamy dwa pliki nagłówkowe - już wiesz do czego pliki nagłówkowe służą. Plik **iostream** dołączamy, aby móc skorzystać z wypisywania na ekran (**cout**) oraz pobierania napisów z klawiatury (**cin**). Natomiast plik **string** dołączamy, aby móc skorzystać z typu **string**, który umożliwia operacje na napisach.

Następnie dołączamy standardową przestrzeń nazw **std**. Mimo, że to już wiesz to powtórzę - dzięki temu skracamy sobie zapis w dalszej części programu.

W kolejnej części programu znajduje się główna funkcja programu main, w której pomiędzy { oraz return 0; wstawiliśmy właściwą treść naszego programu.

Wszystkie linie, które zaczynają się od słowa **cout** służą do wypisywania na ekran. Zauważ, że po każdym cout znajduje się operator **<<**. Zwróć też uwagę, że większość napisów wypisywanych na ekran znajduje się w podwójnych cudzysłowach.

Linia zaczynająca się od **cin** służy do pobrania Twojego imienia z klawiatury. Zwróć uwagę, że po cin znajduje się operator **>>** a następnie słowo imie ale tym razem bez cudzysłowów.

Linia **string imie;** ma sygnalizować, że w naszym programie będziemy chcieli wykorzystać pewien napis. Później okazuje się, że napis pobieramy z klawiatury, a następnie wypisujemy na ekran.

Linia **getchar();** powoduje, że nasz program czeka aż użytkownik naciśnie ENTER. Tak naprawdę getchar to pewna gotowa funkcja, która znajduje się w pliku nagłówkowym iostream. Gdybyśmy na początku nie dołączyli tego pliku do naszego programu, nie moglibyśmy wykorzystać tej funkcji.

**Funkcja ignore()**

Zauważ, że w kodzie programu znajduje się jeszcze jedna linia, o której w ogóle nie wspomniałem:

**cin.ignore();**

Zauważ też, że linię wcześniej znajduje się:

**cin >>imie;**

Otóż linia ze słowem ignore powoduje wyeliminowanie pewnego nieprzyjemnego skutku. Kiedy program poprosił Cię o wpisanie imienia, aby zasygnalizować, że wpisywanie jest już zakończone trzeba było użyć klawisza ENTER. Czy jednak klawisz ENTER wchodzi w skład Twojego imienia? Na pewno nie. Dlatego też klawisz ENTER został niejako zapamiętany, że został wciśnięty, jednak nie został dodany do Twojego imienia.

Co to powoduje? Otóż sprawia to, że w przypadku gdy będziemy chcieli jeszcze raz pobrać jakiś napis z klawiatury mogą pojawić się problemy. Dlatego też użyliśmy funkcji **ignore.** Funkcja ta znajduje się w pliku nagłówkowym **iostream**. Dzięki użyciu tej funkcji sprawiliśmy, że to, że klawisz ENTER został wciśnięty zostanie "zapomniane" i nie będzie żadnych problemów.

Nie pytaj w tym momencie dlaczego jednak zapis linii z funkcją ignore jest taki a nie inny, bowiem na to jest jeszcze dużo za wcześnie. Najlepiej zapamiętaj ten zapis i zapamiętaj, że zawsze kiedy pobieramy coś z klawiatury, warto w następnej linii użyć funkcji ignore - dzięki temu unikniesz problemów na przyszłość.

**Jak działa ten program - wyjaśnienie drugie**

Jeśli nie do końca udało Ci się zrozumieć jak działa ten program, teraz postaram się jeszcze raz przedstawić, co program robi. Zastrzegam jednak, że będzie to bardzo nieformalne tłumaczenie, więc jeśli jesteś już bardziej zaawansowanym użytkownikiem, lepiej omiń następny akapit.

Ogólnie mówiąc program działa tak: najpierw sygnalizuje, że będzie chciał używać napisu. Następnie wypisuje komunikat na ekran z prośbą o podanie imienia. Teraz pobiera Twoje imię z klawiatury oraz zaraz po tym zapobiega skutkowi naciśnięcia przez Ciebie klawisza ENTER. Teraz wypisuje komunikat na ekran o treści "Witaj " wraz z imieniem podanym przez Ciebie wcześniej z klawiatury. Następnie wypisuje kolejny komunikat, w którym gratuluje Ci napisania pierwszego programu. Teraz wypisuje komunikat informacyjny, że należy nacisnąć ENTER, aby zakończyć program i czeka tak długo gdy naciśniesz ENTER. Na samym końcu informuje system operacyjny, że program wykonał się w sposób prawidłowy.

**Wykorzystanie komentarzy do wyjaśnienia**

Poniżej jeszcze jeden przykład jak można wyjaśniać program. Skorzystamy tutaj z komentarzy, bo to one zazwyczaj właśnie służą do wyjaśniania kodu programu.

**#include <iostream> // dla cin, cout, ignore, getchar**

**#include <string> // dla string**

**using namespace std; // wlaczenie standardowej przestrzeni nazw**

**/\***

**Ponizej znajduje sie funkcja main**

**\*/**

**int main ()**

**{**

**string imie ; // bedziemy wykorzystywac napis**

**cout <<"Podaj imie: " ; // wypisanie na ekran**

**cin >>imie ; // pobranie imienia z klawiatury**

**cin.ignore() ; // zignorowanie ENTERa nacisnietego przed momentem**

**cout <<"Witaj "<<imie<<'\n' ; // wypisanie komunikatu**

**cout <<"Gratulacje. To Twoj pierwszy program!"<<'\n'; // wypisanie komunikatu**

**cout <<"Nacisnij ENTER aby zakonczyc"<<'\n'; // wypisanie komunikatu**

**getchar(); //czekanie na nacisniecie klawisza ENTER**

**return 0; // informacja, ze program zadzialal prawidlowo**

**} // koniec funkcji main**

**Czym jest ten "Hello world"**

Na koniec warto dowiedzieć się czym jest "Hello world", o którym być może kiedyś usłyszysz. "Hello world" jest programem, który jest w zasadzie pierwszym programem, jaki użytkownik pisze poznając nowy język programowania. Można by rzec, że jest to taka swoista programistyczna tradycja.

Jak jednak widzisz, pierwszy program był zupełnie inny i znacznie trudniejszy od programu "Hello world". Ale, żeby uspokoić swoje sumienie i Twoją ciekawość, oto przedstawiam Ci program "Hello world" w całej okazałości. Mam nadzieję, że komentarz, co robi ten program jest już w tym momencie zbyteczny.

**#include <iostream>**

**using namespace std;**

**int main ()**

**{**

**cout <<"Hello world"<<'\n';**

**return 0;**

**}**

Jak widzisz program jest bardzo prosty. Jeśli po skompilowaniu i uruchomieniu program tylko miga Ci przed oczami, upewnij się, że znasz różne sposoby uruchamiania programów

.

**Średniki w C++**

Przyjrzyj się teraz kodom. Zauważ, że po większości linijek znajduje się średnik. Zwróć jednak uwagę, że średnik nie znajduje się na końcu każdej linii.

Spróbuj teraz w którymś z przykładowych programów skasować jeden ze średników. Spróbuj skompilować program. Zapewne ujrzysz błąd i program się nie skompiluje. Przywróć początkowy stan programu i teraz dla odmiany spróbuj dodać średnik w którejś z linii, w której średnika nie było. O ile tylko średnik nie został dostawiony w linii pustej lub linii zawierającej któryś z nawiasów klamrowych, programu również nie uda się skompilować.

Jak zatem możesz się już teraz domyślać, średniki znajdujące się na końcach linii nie znajdują się tam przypadkowo. Ich wstawienie lub niewstawienie jest za każdym razem bardzo ważne, bowiem mały błąd może spowodować albo to, że program się nie skompiluje, albo co gorsza, że program się skompiluje, ale będzie działał nie tak, jak autor programu sobie życzył.

Tak naprawdę, nie do końca jest prawdą, że średniki musimy stawiać na końcu niektórych linii.Średniki musimy stawiać na końcu każdej instrukcji.

**Instrukcja w języku C++**

Instrukcję w języku C++ można utożsamiać ze zdaniem. Instrukcja jakby mówi, co ma się zdarzyć np. zadeklaruj zmienną, pobierz zmienną x, wypisz komunikat na ekran, zwróć wartość 0.

Oprócz instrukcji, w języku C++ występują również wyrażenia. Wyrażenia są jakby częściami instrukcji. Każda instrukcja może składać się z kilku lub kilkunastu wyrażeń. Na przykład, jeśli chcemy wypisać sumę liczb 2 i 3, wówczas suma liczb 2 i 3 stanowi wyrażenie, natomiast cała operacja wypisania stanowi instrukcję.

Zasadniczą różnicą między instrukcjami i wyrażeniami jest to, że instrukcja musi zostać zakończona średnikiem. Natomiast wyrażeń nie kończymy średnikami - jeśli tak zrobimy, wówczas kompilator zasygnalizuje błąd.