INFORMATYKA

PODSTAWY JĘZYKA C

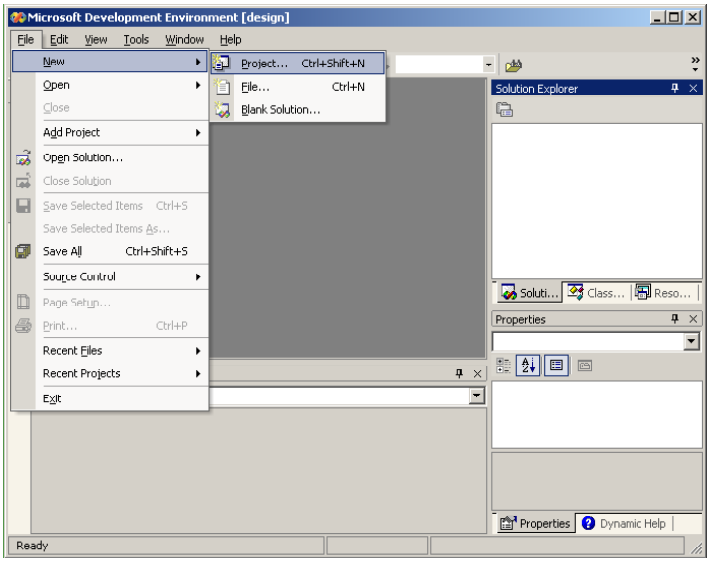
CEL LABORATORIUM

Celem zajęć jest zapoznanie uczestniczki/uczestnika z podstawami języka C. Najpierw utworzony zostanie projekt w programie Visual Studio .NET 2003. Następnie przedstawiony i omówiony zostanie kod programu „witaj świecie". W trakcie zajęć przedstawiony zostanie także sposób pracy ze zmiennymi i instrukcjami warunkowymi.

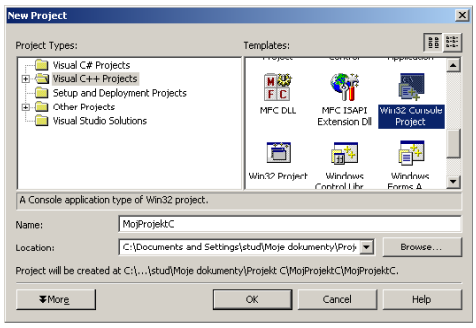
TWORZENIE NOWEGO PROJEKTU W ŚRODOWISKU VISUAL STUDIO

Uruchom program Oracle **VM VirtualBox** i maszynę wirtualną wskazaną przez osobę prowadzącą zajęcia. Jeżeli prace wykonujesz na maszynie wirtualnej SO_BD_IO, otwórz menu „`Start`", najedź kursorem na pozycję „`Wszystkie programy`", a następnie na „`Microsoft Visual Studio .NET 2003`". Możesz również skorzystać z własnego komputera lub z innych środowisk obecnych na komputerach w laboratorium. Upewnij się, że Twoje środowisko obsługuje języki C i C++.

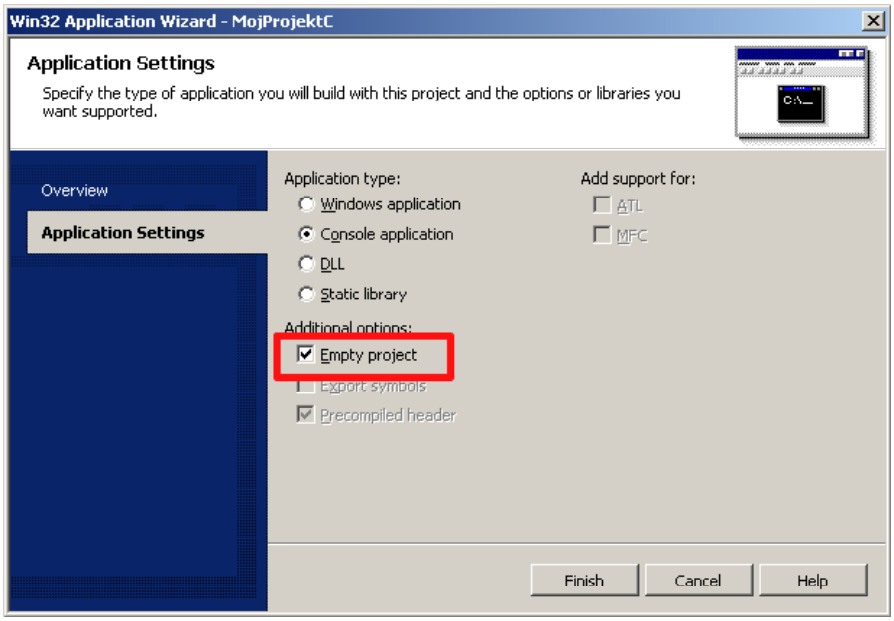
W oknie, które się po chwili uruchomi, w pasku menu znajdź polecenie „`File`" i z menu kontekstowego wybierz `New` → `Project…` , tak, jak pokazano na rysunku poniżej. Jeżeli pracujesz na innej maszynie wirtualnej lub systemie i nie możesz samodzielnie znaleźć programu Visual Studio, zapytaj o pomoc prowadzącego.



W oknie, które się otworzy, jako typ ustaw „`Visual C++ Projects`", po czym z listy dostępnych szablonów wybierz „`Win32 Console Project`". Wpisz nazwę dla projektu i zwróć uwagę, w jakim katalogu zostanie utworzony. Katalog okaże się przydatny na końcu zajęć, gdy zajdzie konieczność skopiowania projektu w celu przechowania lub wysłania prowadzącemu. Po wykonaniu tych czynności, wciśnij przycisk „`OK`".

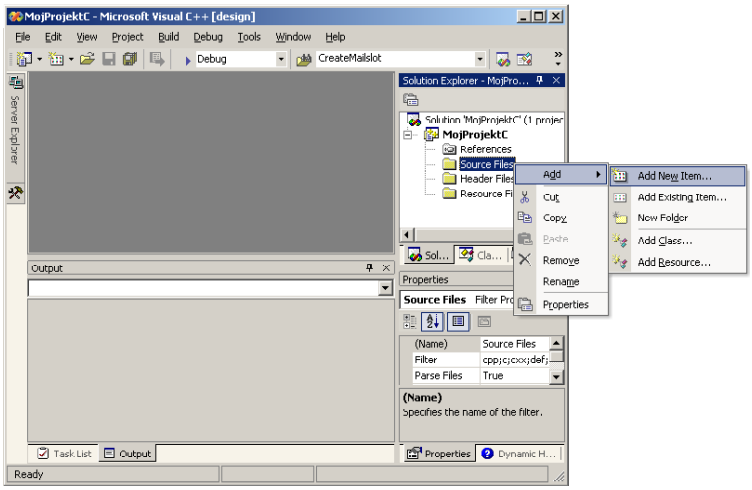


Otworzy się okienko kreatora. Zawiera ono podsumowanie ustawień nowotworzonego projektu. Przejdź do zakładki „Application Settings” i zaznacz „Empty Project”. Następnie zatwierdź przyciskiem „Finish”.

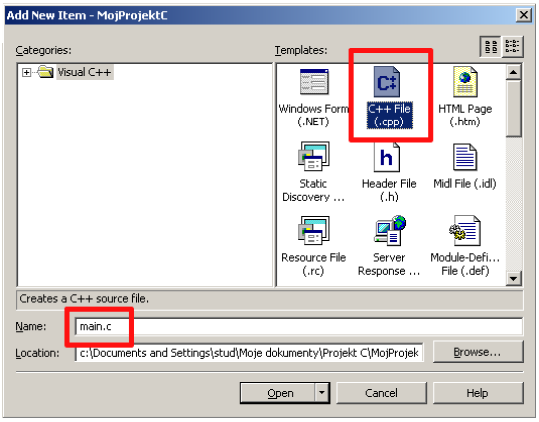


TWORZENIE NOWEGO PLIKU

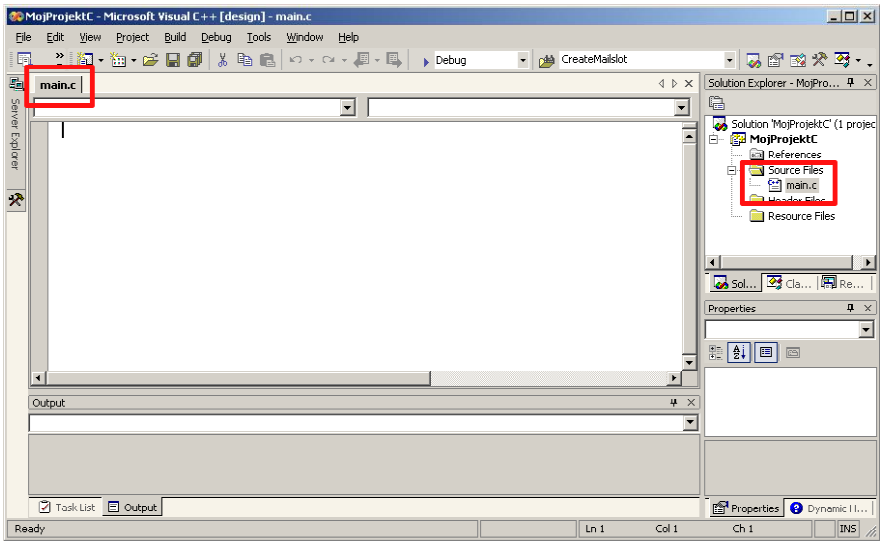
Żeby móc rozpocząć pracę, potrzebujemy stworzyć plik, który będzie zawierał kod, który będziemy pisać. Projekty w Visual Studio mogą składać się z wielu plików, ale w obecnej chwili, potrzebujemy wyłącznie jeden. Po prawej stronie znajduje się Solution Explorer. Kliknij **prawym** przyciskiem myszki na katalogu „Source Files”, a następnie z menu kontekstowego wybierz Add →Add New Item…



Następnie, w oknie, które się otworzy, znajdź i kliknij szablon „`C++ File (.cpp)`”, a następnie nadaj mu nazwę „`main.c`”[1]. Gdy to zrobisz, wciśnij przycisk „Open”.



Nowoutworzony plik zostanie automatycznie dodany do projektu i otworzony w głównym oknie programu tak, jak pokazuje do poniższy zrzut ekranu. Wszystko jest teraz gotowe, możemy zacząć naukę języka C.



INNE ŚRODOWISKA PROGRAMISTYCZNE

Visual Studio nie jest jedynym środowiskiem programistycznym, które obsługuje język C. Istnieją inne, popularne i darmowe środowiska, z których możesz korzystać w domu lub pracując na swoim komputerze w trakcie zajęć:

- Eclipse (https://www.eclipse.org/downloads)
- NetBeans (https://netbeans.apache.org/download/index.html)
- Code::Blocks (https://www.codeblocks.org/downloads/)
- Visual Studio Code (https://code.visualstudio.com/)
- Dev C++ (http://www.bloodshed.net/)

[1] Nazwa, w tym „c” na jej końcu, jest istotna. Nie zmieniaj jej.

Istnieją również środowiska online, które nie wymagają instalacji oprogramowania, ale są wolniejsze. Przykładem takiego środowiska może być Online GDB:

- OnlineGDB: https://www.onlinegdb.com/online_c_compiler

Pracując z innym środowiskiem, miej świadomość, że różnić się będą komunikaty błędów, a prowadzący zajęcia może nie mieć doświadczenia z pracą w danym środowisku, więc ciężej będzie mu pomóc Ci rozwiązać problem. Wybierając środowisko miej również na uwadze to, żeby pozwalało pracować z wieloma plikami. Będzie to wymagane w dalszej części semestru, a nie wszystkie środowiska (szczególnie te online), taką możliwość oferują.

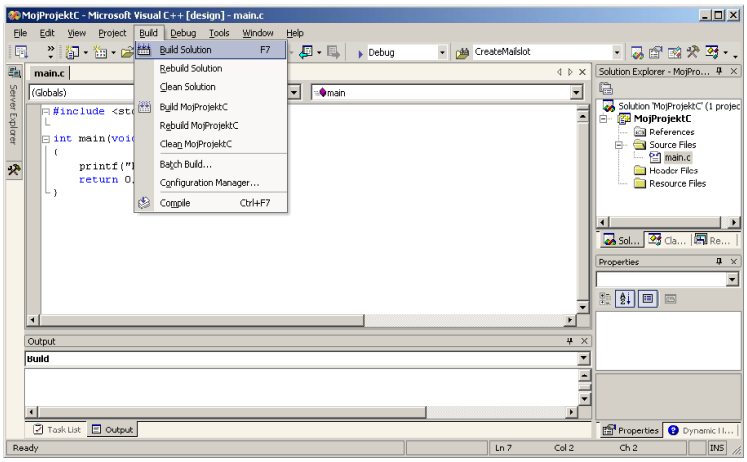
WITAJ ŚWIECIE!

Naukę programowania zazwyczaj zaczyna się od programu zwanego potocznie „witaj świecie!”. Celem tego programu jest uruchomić się i wypisać na ekranie napis „witaj świecie!” („Hello world!”). W języku C, program ten ma następującą postać:

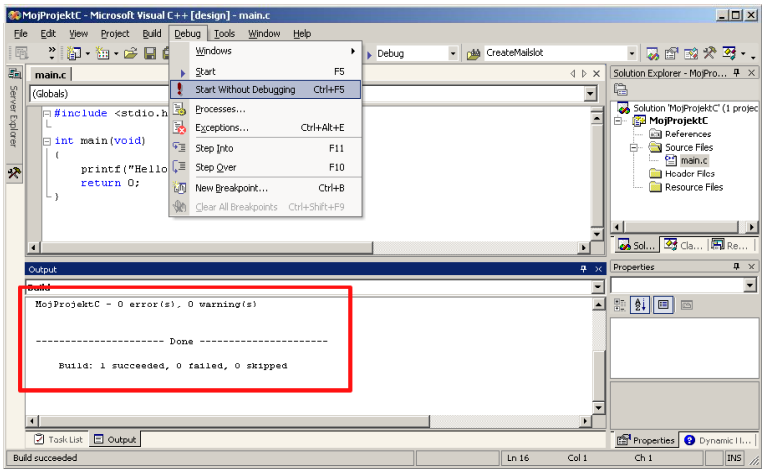
```
#include <stdio.h>

int main(void)
{
  printf("Hello world!");
  return 0;
}
```

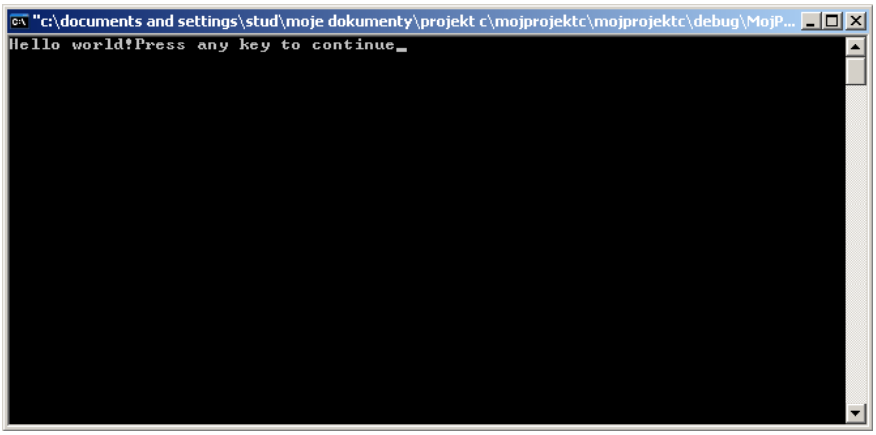
Przepisz kod do pliku main.c, zapisz go, a następnie z menu wybierz `Build` → `Build Solution (F7)`:



Jeżeli kod jest **poprawny składniowo**, na dole, w oknie „Wyjście”, pokaże się komunikat o braku błędów i poprawnym utworzeniu programu. Jeżeli tak nie jest, kod musi zostać poprawiony, zanim będzie możliwe uruchomienie programu. Jeżeli operacja się powiodła, z menu wybierz „`Debug`”, a następnie „`Start without Debugging`”:



Jeżeli kod jest **poprawny logicznie**, na ekranie powinna się pojawić konsola systemu Windows, zawierająca „`Hello world!`” oraz komunikat „naciśnij dowolny klawisz, aby kontynuować”:



W JAKI SPOSÓB Z KODU POWSTAJE PROGRAM?

Napisaliśmy nasz pierwszy program komputerowy. Warto zastanowić się jak wygląda proces tłumaczenia kodu na program (czyli zrozumiały dla systemu ciąg zer i jedynek). Istnieją dwa rozwiązania.

Kod może być tłumaczony (**interpretowany**) za pomocą specjalnego programu, zwanego **interpreterem**. Interpreter analizuje kod programu linijka po linijce i **wykonuje** zapisane w nim instrukcje. Żeby kod napisany na jednym komputerze, działał na innym, użytkownik musi skopiować lub zainstalować u siebie interpreter, a także posiadać sam kod. Kod jest przechowywany w formie jawnej, a więc każda osoba, która zna odpowiedni język programowania, może go przeczytać czy skopiować i użyć we własnym programie. Z tym faktem związane są kwestie bezpieczeństwa i własności intelektualnej.

Alternatywnym podejściem jest **kompilacja**. W tym przypadku, kod zazwyczaj jest wstępnie przetwarzany przez program zwany **preprocesorem** (od angielskiego „*preprocess*”), a następnie przez kolejny program (zwany **kompilatorem**), który przetwarza kod źródłowy na kod niskopoziomowy, to znaczy taki, który operuje bezpośrednio na instrukcjach procesora. Następnie wchodzi trzeci program, zwany **assemblerem**, który tłumaczy kod niskopoziomowy na sekwencję zer i jedynek, zwaną kodem maszynowym. Na ostatnim etapie, uruchamiany jest program, zwany **linkerem**, który łączy wyprodukowane fragmenty kodu w jeden program.

Języki C i C++, których będziemy się uczyć w trakcie trwania semestru, są językami kompilowanymi. Ma to swoje wady i zalety. Zaletą jest oczywiście to, że nie musimy dostarczać z naszym programem interpretera,

a sam kod jest nieczytelny i możliwe jest wyłącznie łatwe odczytanie kodu niskopoziomowego. Wadą jest fakt, że na etapie kompilowania, program zostaje przystosowany do konkretnego **rodzaju** procesora. Program skompilowany na komputerze, nie będzie działał na telefonie, gdyż urządzenia te różnią się rodzajem procesora – komputery operują na architekturze x86, a telefony na procesorach o architekturze ARM.

WYJAŚNIENIE KODU PROGRAMU

Przeanalizujmy kod naszego programu. Pierwszą linijką jest „`#include <stdio.h>`". Każda linijka, która zaczyna się od znaku hash („#"), nazywamy **instrukcją preprocesora**. W dużym uproszczeniu, instrukcje preprocesora mówią, że przed wykonaniem programu, należy wykonać pewne czynności, które ten kod przygotują. Instrukcja `include` (ang.: „*dołącz*"), służy do dołączenia plików do naszego programu.

W tym wypadku dołączany jest plik `stdio.h`. Nazwa „stdio" jest skrótem od angielskiego zwrotu „*standard input/output*", który oznacza „*standardowe metody wejścia/wyjścia*", które są w tym pliku zawarte. Plik ten jest dołączony do każdego kompilatora, dlatego możemy z niego skorzystać.

Kolejna linijka zawiera „`int main(void)`". Jest to funkcja główna (ang. „*main*") naszego programu. Czym jest „`int`" oraz „`void`" i dlaczego znajdują się przed i po słowie `main` oraz po co są nawiasy, dowiesz się w trakcie tych i kolejnych zajęć laboratoryjnych. W tej chwili najbardziej istotne jest zrozumienie, że każdy program **musi zawierać dokładnie jedną** funkcję `main` i **musi mieć ona taką, a nie inną postać**.

Następnym elementem jest znak klamry „{", po którym następuje dwie linijki kodu i drugi znak klamry „}". Klamry wyznaczają obszar funkcji `main`. Obszar kodu zawarty pomiędzy klamrami, nazywamy **blokiem kodu**.

Po klamerce znajdziemy linijkę „`printf("Hello world!");`". Jest to pierwsza **instrukcja** naszego programu. Słowo „`printf`" (od ang. „*print formatted*", czyli „*wydrukuj sformatowane*") to nazwa funkcji służącej do wypisania pewnego ciągu znaków na ekranie konsoli. Po nazwie funkcji znajdują się nawiasy, pomiędzy którymi jest zawarty ciąg znaków, który należy wypisać. Zwróć uwagę na dwie rzeczy. Ciąg znaków jest otoczony cudzysłowami, a na końcu linijki znajduje się średnik. Obie te rzeczy są wymagane – **cudzysłowy znajdziemy zawsze wokół każdego ciągu znaków**, a **średnik jest wymagany po każdej instrukcji**.

Ostatnią linijką kodu jest „`return 0;`". Angielskie słowo „*return*" oznacza „*zwróć*" i dokładnie temu służy. W programowaniu przyjęło się, że jeżeli program zakończy się z wynikiem „0", oznacza to, że wykonał się bezbłędnie. Jako że jest to instrukcja, należy na jej końcu umieścić średnik.

ZADANIE 1

Wykonaj następujące modyfikacje kodu. Po każdej modyfikacji kliknij `Build` → `Build Solution` i zwróć uwagę na błędy, które się pojawiają. Zapoznanie się z treścią błędów, znacznie przyspieszy Ci dalszą przygodę z programowaniem. **Po zapoznaniu się z treścią błędu, przed przejściem do kolejnego podpunktu, przywróć kod do wersji poprawnej**.

a) Zmień w pierwszej linijce kodu „`stdio.h`" na „`stdioo.h`".
b) Zmień nazwę funkcji „`main`", na „`mian`".
c) Usuń klamrę otwierającą blok funkcji `main`.
d) Usuń dowolny jeden z cudzysłowów wokół „Hello world".
e) Usuń nawias zamykający wywołanie funkcji `printf`.
f) Zmień „`printf`" na „`prinft`".
g) Usuń średnik z końca dowolnej linijki zawierającej instrukcję.
h) Przesuń linijkę zawierającą wywołanie funkcji `printf`, pod klamrę kończącą blok funkcji `main`.

ZNAK NOWEJ LINII

Zauważyłaś/zauważyłeś zapewne, że cały tekst wypisany jest na konsoli w jednej linijce. Jest to poprawne, ale nieczytelne. Istnieje wiele wyjść z tej sytuacji. Najprostszą jest wstawienie spacji po wykrzykniku:

```
printf("Hello world! ");
```

Wprowadź omawianą zmianę, skompiluj (zbuduj) program ponownie i uruchom. Zaobserwujesz, że pomiędzy „witaj świecie", a „wciśnij dowolny klawisz", jest teraz przerwa. Teraz tekst wygląda lepiej. Najlepiej jednak by było, gdyby komunikat o wciśnięciu dowolnego klawisza, był w osobnej linii. To też jest oczywiście możliwe.

Żeby przejść do kolejnej linii konsoli, wymagane jest wypisanie specjalnej sekwencji znaków: `\n`.[2] Znak „\" nazywamy *backslashem* („tylnym cięciem") i nie należy go mylić z „/" (*slashem, cięciem*). Dodaj tę sekwencję na końcu tekstu, skompiluj program ponownie i uruchom. Zaobserwujesz, że „witaj świecie" i „wciśnij dowolny klawisz", są teraz w osobnych liniach konsoli.

```
printf("Hello world!\n");
```

Możesz oczywiście wstawić znak nowej linii w dowolnym miejscu ciągu znaków, a także w dowolnej liczbie powtórzeń:

```
printf("Hello\nworld!\n\n\n\n");
```

ZMIENNE

Na poprzednim laboratorium mówiliśmy, że zmiennymi nazywamy specjalne miejsca w pamięci komputera (na dysku twardym lub w pamięci ulotnej), którym przypisano nazwę i można się do nich odnieść. W języku C, istnieje jeszcze jeden wymóg dotyczący zmiennych – w momencie utworzenia, muszą mieć one ściśle określony **typ**. Poniżej znajdziesz tabelkę podstawowych typów zmiennych:

Typ	Skąd taka nazwa?	Opis (dla systemów 32-bitowych)
`int`	Ang. *„integer"*, liczba całkowita	Liczba całkowita (32 bity / 4 bajty)
`float`	Ang. *„floating point"*, zmienny przecinek	Liczba zmiennoprzecinkowa w systemie IEEE 754 (32 bity / 4 bajty)
`double`	Ang. *„double precision"*, podwójna precyzja	Liczba zmiennoprzecinkowa podwójnej precyzji w systemie IEEE 754 (64 bity / 8 bajtów)
`char`	Ang. *„character"*, znak	Pojedynczy znak w kodowaniu ASCII, zapisany na 8 bitach (jednym bajcie)
`void`	Ang. *„void"*, pustka, otchłań, nic	Specjalny typ informujący o tym, że funkcja nie zwraca lub nie pobiera zmiennych żadnego typu

Moment utworzenia zmiennej nazywamy **deklaracją**. Instrukcja deklaracji zmiennej w języku C, przyjmuje następującą postać:

```
typ nazwa;
```

Na przykład:

[2] Na systemach z rodziny uniksowej, z których język C się wywodzi, sekwencja ta przyjmie postać `\r\n`, gdzie `\r`, oznacza powrót do początku wiersza (*„powrót karetki"*, ang. *„carriage return"*).

```
int liczba;
```

Nazwa zmiennej może być dowolna, ale musi spełniać pewne ograniczenia. Może zawierać wyłącznie litery alfabetu angielskiego (zarówno duże jak i małe), cyfry i znak „_”. Nie może zaczynać się cyfrą. Nie może być słowem kluczowym języka (np. nie można nazwać zmiennej „int”). Nazwa zmiennej musi być unikalna w ramach danego bloku kodu.

Żeby przypisać wartość zmiennej, należy wykorzystać znak „=” tak, jak pokazano poniżej:

```
int liczba2;
liczba2 = 5;
```

Można także połączyć deklarację i nadanie wartości w jedną instrukcję. Taką instrukcję nazywamy **definicją zmiennej** i ma ona następującą postać:

```
typ nazwa = wartość;
```

Na przykład:

```
float liczbaNiecalkowita = 10.29;
```

Na zmiennych można wykonywać operacje arytmetyczne, takie jak dodawanie, odejmowanie, mnożenie, dzielenie i przypisanie. Żeby było to możliwe, zmienną należy najpierw zadeklarować lub zdefiniować:

```
float a = 1.14;
float b = 2.53;
float c = 1.0/89.0;
double delta;
delta = b*b – 4*a*c;
```

W języku C, definicje i deklaracje zmiennych muszą wystąpić przed wszelkimi innymi instrukcjami w danym bloku kodu. Oznacza to, że następujący kod jest niepoprawny:

```
float a;
a = 1.14;
double delta; // BLAD! Zmienna zadeklarowana po wykonaniu instrukcji!
```

Natomiast ten kod jest poprawny, gdyż delta jest zadeklarowana w osobnym bloku kodu:

```
float a;
a = 1.14;
{
        double delta; // W porzadku, mamy osobny blok kodu.
}
```

ZADANIE 2 Ⓟ

Zadeklaruj zmienną typu znakowego. Przypisz do niej wartość ‘a’.

ZADANIE 3

Zdefiniuj zmienną typu całkowitoliczbowego, nazywającą się przykladowaNazwaZmiennej.

NAZYWANIE ZMIENNYCH

Wielu początkujących programistów, nazywając zmienne, stosuje bardzo krótkie i zwięzłe nazwy – `x, y, a, b, n1, n2` i tym podobne. Każda z tych nazw jest oczywiście prawidłowa. Ale to nie znaczy, że jest dobra. Postaraj się, żeby każda zmienna, którą tworzysz, miała nazwę, która oddaje cel jej istnienia. Jeżeli będziesz mieć przerwę od pracy nad swoim programem i wrócisz do niego po miesiącu, instrukcja „`t = 5200;`”, absolutnie nic Ci nie będzie mówić. W przeciwieństwie do instrukcji „`maksymalnaTemperaturaReaktora = 5200;`”.

MODYFIKATORY TYPU

Do typów możliwe jest dodanie modyfikatorów. W języku C istnieją 4 modyfikatory:

Modyfikator	Znaczenie
`long`	Zwiększa liczbę bitów przeznaczoną do przechowywania wartości
`short`	Zmniejsza liczbę bitów przeznaczoną do przechowywania wartości
`signed`	Wartości przechowywane w zmiennej są traktowane jako wartości ze znakiem
`unsigned`	Wartości przechowywane w zmiennej są traktowane jako wartości bez znaku (a więc wyłącznie dodatnie)

Modyfikatory można łączyć, o ile nie wykluczają się. W przypadku, gdy stosowany jest modyfikator, powinien on wystąpić przed typem, co sprawia, że deklaracja przyjmuje postać:

```
modyfikator typ nazwa;
```

Przykład poprawnej definicji:

```
unsigned long int duzaDodatnia = 4294967295;
```

ŁĄCZENIE OPERACJI Z PRZYPISANIEM, INKREMENTACJA I DEKREMENTACJA

Dla przyspieszenia pracy programistów, język C oferuje kilka specjalnych operatorów, które reprezentują najczęściej wykonywane kombinacje operacji. Nie musisz ich stosować, ale możesz się z nimi spotkać w literaturze lub przeglądając czyiś kod. Lepiej się więc z nimi zapoznać.

Operator	Działanie	Przykład

`+= -= *= /= &= |= ~= %=`	Wywołuje operacje arytmetyczną/binarną (dodawanie, odejmowanie, mnożenie, dzielenie, i, lub, negacja, reszta z dzielenia) lewego argumentu z prawym, po czym przypisuje wynik do lewego argumentu.	`a+=b // a=a+b`
		`a&=b // a=a&b`
		`a%=b // a=a%b`

`++x`	Zwiększa wartość x o 1, po czym zwraca wartość x+1	`a=++b // a=b+1, b=b+1`
`--x`	Zmniejsza wartość x o 1, po czym zwraca wartość x-1	`a=--b // a=b-1, b=b-1`
`x++`	Zwraca wartość x, po czym zwiększa wartość x o 1	`a=b++ // a=b, b=b+1`
`x--`	Zwraca wartość x, po czym zmniejsza wartość x o 1	`a=b-- // a=b, b=b-1`

WYPISYWANIE WARTOŚCI ZMIENNYCH

Nauczyliśmy się tworzyć zmienne, nadawać im wartość i wykonywać podstawowe operacje arytmetyczne. To wszystko jest jednak nieistotne, jeżeli nie dowiemy się, jak wypisywać ich wartość na ekranie i podawać za pomocą klawiatury. Zacznijmy od wypisywania na ekranie. W pierwszej aplikacji, którą opracowaliśmy, do wypisywania na ekranie używaliśmy funkcji `printf`.

Ta sama funkcja służy do wypisywania na ekranie nazw zmiennych. Wystarczy przekazać do niej odpowiednie argumenty. Pierwszym argumentem powinien być ciąg znaków, informujący komputer, jakiego typu jest zmienna i jak ją sformatować, a drugim argumentem powinna być sama zmienna. Stąd, pierwszy argument funkcji printf, nazywamy **ciągiem formatującym**. Poniżej znajduje się tabelka, w której zawarto typy zmiennych i przypisane im znaczniki formatowania.

Typ	Formatowanie
`char`	`%c`
`int`	`%i`
`float`	`%f`
`double`	`%lf`

Dla przykładu, **przepisz** następujący kod i zaobserwuj jego działanie:

```
#include <stdio.h>

int main(void)
{
  int ulubiona = 42;
  printf("Moja ulubiona liczba to: %i!\n", ulubiona);
  return 0;
}
```

Możliwe jest wypisanie wartości kilku zmiennych jednocześnie, w jednej instrukcji `printf`:

```
#include <stdio.h>

int main(void)
{
  int ulubiona = 42;
  float pi = 3.14;

  printf("Moja ulubiona liczba to: %i, a pi = %f!\n", ulubiona, pi);
  return 0;
}
```

POBIERANIE DANYCH Z KLAWIATURY

Do pobierania danych z klawiatury, służy siostrzana instrukcja `printf`, która nazywa się `scanf` (od angielskiego „*scan formatted*”). Używa jej się analogicznie jak `printf`, z tą różnicą, że samą zmienną przekazujemy poprzedzając jej nazwę et (znakiem *ampersand*, „`&`”). Znaczenie tego znaku poznasz podczas dalszej nauki języka. W tej chwili **zapamiętaj**, że w `printf` nie używamy ampersandu, a w `scanf` jest on wymagany.

ZADANIE 4

Przepisz poniższy kod, skompiluj i uruchom:

```
#include <stdio.h>

int main(void)
{
  int ulubiona = 0;
  printf("Jaka jest Twoja ulubiona liczba? ");
  scanf("%i", &ulubiona);
  printf("Twoja ulubiona liczba to %i!\n", ulubiona);
  return 0;
}
```

INSTRUKCJA WARUNKOWA

Kolejnym elementem języka C jest instrukcja warunkowa `if-else` (ang. *„if” – „jeżeli”*, „else” – „w przeciwnym wypadku”). Jej działanie jest takie, jakie miała instrukcja warunkowa, którą poznaliśmy podczas omawiania algorytmów. Jeżeli warunek jest prawdziwy, wykonywane są instrukcje zawarte w pierwszym bloku kodu (tym po słowie kluczowym `if`). W przeciwnym wypadku (czyli gdy warunek nie jest prawdziwy), wykonywane są wyłącznie instrukcje zawarte w drugim bloku kodu (tym po słowie kluczowym `else`). Instrukcja warunkowa ma następującą postać:

```
if(warunek)
{
        // wykonywane, jeżeli warunek spelniony
}
else
{
        // wykonywane, jeżeli warunek nie jest spelniony
}
```

Warunek uznajemy za **spełniony** (lub prawdziwy), gdy jego wartość jest **różna od zera**. W przypadku liczb, warunkiem może być więc sama liczba lub jej stosunek do innej zmiennej czy stałej. Stosunek pomiędzy liczbami można zbadać następującymi operatorami:

Operator	Działanie	Przykład prawdy
`==`	Prawda, jeżeli wartości są **równe**	`10 == 8+2`
`!=`	Prawda, jeżeli wartości **nie są równe**	`2+2 != 5`
`<`	Prawda, jeżeli pierwsza wartość jest **mniejsza** od drugiej	`3 < 4`
`<=`	Prawda, jeżeli pierwsza wartość jest **mniejsza lub równa** drugiej	`6 <= 9-3`
`>`	Prawda, jeżeli pierwsza wartości jest **większa** od drugiej	`7 > 1`
`>=`	Prawda, jeżeli pierwsza wartość jest **większa lub równa** drugiej	`3*4 >= 2*6`

Warunki można ze sobą łączyć. Służą do tego operatory logicznej negacji, alternatywy i koniunkcji. Z wyjątkiem operatora negacji, operatory te wyglądają bardzo podobnie do operatorów binarnych. Warto zestawić je w jednym miejscu i opisać:

Operator	Działanie
~	Operator negacji **binarnej**. Odwraca wartości bitów zmiennej.
!	Operator negacji **logicznej**. Zmienia prawdę na fałsz, a fałsz na prawdę.
&	Operator koniunkcji **binarnej**. Wynikiem jest wartość, której poszczególne bity są jedynkami tylko wtedy, jeżeli dwie wartości składowe również miały w tym miejscu jedynki.
&&	Operator koniunkcji **logicznej**. Wynikiem jest prawda, jeżeli warunki składowe były prawdą, a fałsz w przeciwnym wypadku.
|	Operator alternatywy **binarnej**. Wynikiem jest wartość, której poszczególne bity są jedynkami wtedy, gdy były jedynką w dowolnej z wartości składowych.
||	Operator alternatywy **logicznej**. Wynikiem jest prawda, jeżeli którykolwiek warunek składowy był prawdziwy.

Należy tutaj zaznaczyć, że operatory mają swój priorytet. Zostaną one wykonane w następującej kolejności: najpierw `!`, następnie `~`, `&`, `|`, `&&` i na końcu `||`. Jeżeli potrzebna jest inna kolejność, należy zastosować odpowiednio umieszczone nawiasy.

ZADANIA DO SAMODZIELNEGO WYKONANIA

ZADANIE 5

Napisz w języku C program, który prosi użytkownika o podanie wartości dwóch liczb zmiennoprzecinkowych. Następnie wypisuje na ekranie komunikat o treści *„[x] jest wieksze od [y]”*, a w miejsce [x] i [y], wstawia odpowiednie wartości, które użytkownik wcześniej podał z klawiatury. Program skompiluj, uruchom i przetestuj.

ZADANIE 6 Ⓟ

Napisz w języku C program, który prosi użytkownika o podanie wartości całkowitoliczbowej. Jeżeli liczba jest w zakresie od 5, do 17 włącznie, program powinien wypisać na ekranie *„Dzien dobry!”*. Jeżeli liczba jest w zakresie od 18 do 23 włącznie, program powinien wypisać na ekranie *„Dobry wieczor!”*. W przeciwnym wypadku program powinien wypisać na ekranie *„Dobranoc!”*. Program skompiluj, uruchom i przetestuj.

ZADANIE 7 Ⓟ (NAGRADZANE DWOMA PLUSAMI)

Napisz w języku C program, który prosi użytkownika o podanie wartości trzech liczb zmiennoprzecinkowych, współczynników równania kwadratowego („a”, „b” i „c”). Następnie program ma obliczyć (za pomocą „delty”) i wypisać na ekranie rozwiązanie (gdy delta=0) lub rozwiązania (gdy delta>0) równania opisanego takimi współczynnikami. Jeżeli delta jest mniejsza niż 0, wypisz na ekranie „Rownanie nie ma rozwiazan!”. Program skompiluj, uruchom i przetestuj.