Języki i paradygmaty programowania 1 – studia stacjonarne 2025/26

Lab 1. Wstęp do programowania w C. Zintegrowane środowisko programistyczne Visual Studio.

Podstawowe pojęcia:

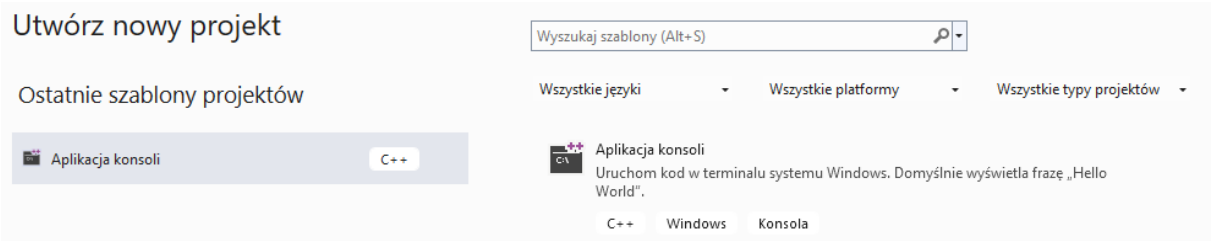
1. ***C*** *– imperatywny, strukturalny język programowania wysokiego poziomu stworzony na początku lat siedemdziesiątych XX w. przez Dennisa Ritchiego do programowania systemów operacyjnych i innych zadań niskiego poziomu.*
2. ***Microsoft Visual Studio*** *– zintegrowane środowisko programistyczne (IDE) firmy Microsoft. Jest używane do tworzenia oprogramowania konsolowego oraz z graficznym interfejsem użytkownika.*
3. ***Kompilator*** *– program służący do automatycznego tłumaczenia kodu napisanego w jednym języku (języku źródłowym) na równoważny kod w innym języku (języku wynikowym). Proces ten nazywany jest kompilacją. W informatyce kompilatorem nazywa się najczęściej program do tłumaczenia kodu źródłowego w języku programowania na język maszynowy. Niektóre z nich tłumaczą najpierw do języka asemblera, a ten na język maszynowy jest tłumaczony przez asembler.*
4. ***Konsolidator*** *(ang. linker) lub program konsolidujący to jeden z programów składowych kompilatora. Konsolidator w trakcie procesu konsolidacji łączy zadane pliki obiektowe i biblioteki statyczne tworząc w ten sposób plik wykonywalny.*
5. ***Pliki nagłówkowe*** *– przy kompilatorze języka C i pochodnych (C++ itp.) pliki źródłowe o rozszerzeniu "h" (w C++ powinno się dla odróżnienia stosować "hpp", ale w praktyce najczęściej spotykane jest nadal rozszerzenie "h") zawierające opis interfejsu modułu: deklaracje zmiennych, funkcji, klas i innych struktur danych. Używa się ich po to, by nie trzeba było przy każdej najmniejszej zmianie w implementacji jednego modułu rekompilować wszystkich innych odwołujących się do niego.*
6. ***Preprocesor*** *– program komputerowy, którego zadaniem jest przetworzenie kodu źródłowego, w sposób określony przez programistę za pomocą dyrektyw preprocesora, na kod wyjściowy – tak przetworzony kod źródłowy poddawany jest następnie analizie składniowej, kompilacji, a w końcu konsolidacji. Preprocesor jest najczęściej zintegrowany z kompilatorem języka programowania.*
7. ***Debugger*** *– program komputerowy służący do dynamicznej analizy innych programów, w celu odnalezienia i identyfikacji zawartych w nich błędów, zwanych z angielskiego bugami (robakami). Proces nadzorowania wykonania programu za pomocą debuggera określa się mianem debugowania.*

   Podstawowym zadaniem debuggera jest sprawowanie kontroli nad wykonaniem kodu, co umożliwia zlokalizowanie instrukcji odpowiedzialnych za wadliwe działanie programu. Współczesne debuggery pozwalają na efektywne śledzenie wartości poszczególnych zmiennych, wykonywanie instrukcji krok po kroku czy wstrzymywanie działania programu w określonych miejscach. Debugger jest standardowym wyposażeniem większości współczesnych środowisk programistycznych.

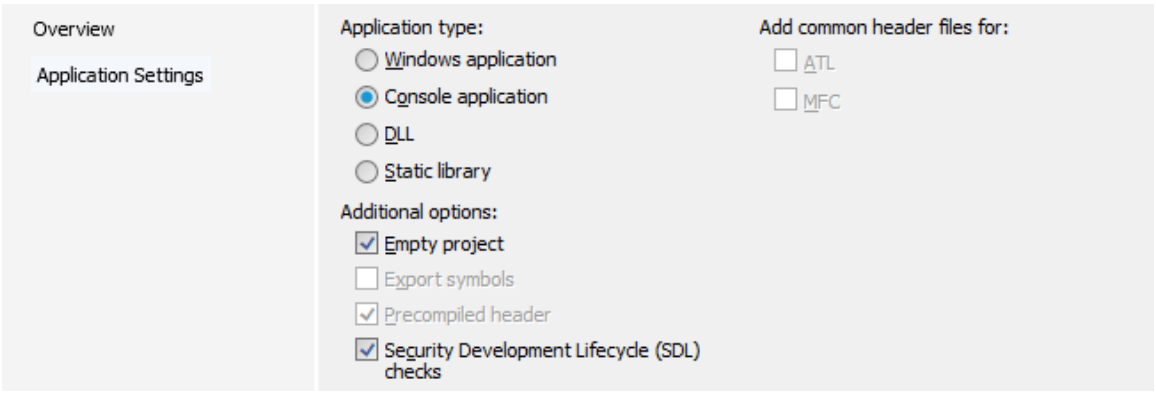
   Debuggery posiadają również wady – symulacja działania kodu nie jest idealnym odtworzeniem wykonania tego kodu w warunkach normalnych. Wobec tego debuggery mogą nie wykrywać bugów niezależnych bezpośrednio od treści badanego programu.

Laboratoria:

1. Uruchom program VS. Utwórz nowy projekt dla języka C/C++ - aplikację konsolową. Nazwa projektu może być dowolna (bez polskich znaków), np. zgodna z numerem laboratorium.

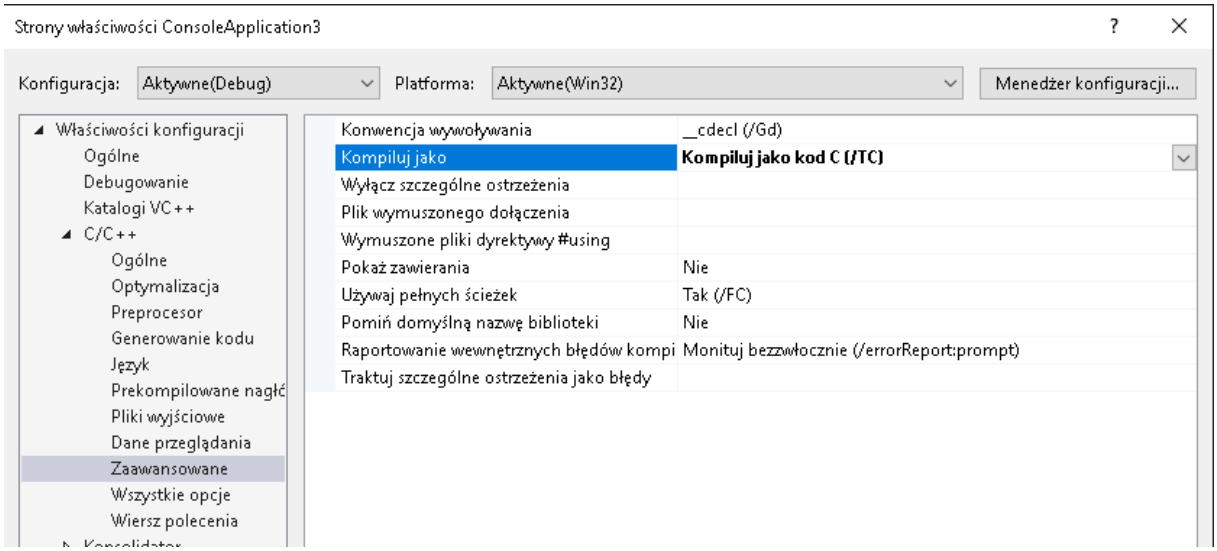


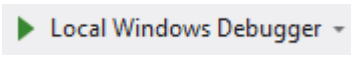
Starsze wersje VS. W ustawieniach kreowania nowego projektu wybierz opcję „Empty project” .



Wybór projektu z zaznaczonym „Precompiled header” tworzy projekt z dołączonym plikiem nagłówkowym Stdafx.h, który przyspiesza proces kompilacji w przypadku bardziej rozbudowanych projektów. Plik ten dołącza najczęściej wykorzystywane biblioteki, które zostają przez kompilator przeprocesowane w kod pośredni.

2. We właściwościach projektu ustaw: Kompiluj jako kod C



3. Piszemy pierwszy program. Przyjęło się, że pierwszy program wyświetla napis: „Hello world!” Tak też będzie w naszym przypadku. Przekopiuj poniższy kod i przekompiluj:  lub F5.

```
#include <stdio.h>   /* Dyrektywa prekompilatora dołączająca bibliotekę:
                        Standard Input/Output, czyli standardowe wejście-wyjście */

int main(int argc, char* argv[]) // Główna funkcja programu (typ int)
{
    puts("Hello world!");       // Funkcja wyświetlająca ciąg znaków.
                                // Dodaje znak nowej linii.
    return 0; // Instrukcja kończąca wykonywanie funkcji. Zwraca wartość 0.
}
```

Jaki jest wynik programu? Napraw dodając w odpowiednim miejscu funkcję: `getchar();`
Korzystając z dokumentacji MSDN sprawdź sposób działania funkcji `getchar();` (https://msdn.microsoft.com/pl-pl/library/x198c66c.aspx).
Zastąp funkcję `puts()` funkcją `printf()`.

4. Komentarze: w języku C możliwe są dwa sposoby komentowania:
   I. Komentarz ograniczający się do jednej linii rozpoczynający się od **//**, np.
   // To jest komentarz pojedynczy
   II. Komentarz wieloliniowy rozpoczynający się od **/***, a kończący ***/**, np.
   /* To jest
   Komentarz wieloliniowy */
   UWAGA! Komentarz typu // nie jest zgodny ze standardem ANSI C
5. *Nazwy zmiennych, stałych i funkcji.*
Identyfikatory, czyli nazwy zmiennych, stałych i funkcji mogą składać się z liter (bez polskich znaków), cyfr i znaku podkreślenia z tym, że nazwa taka nie może zaczynać się od cyfry. Nie można używać nazw zarezerwowanych.

Przykłady błędnych nazw:

- *2liczba (nie można zaczynać nazwy od cyfry)*
- *moja funkcja (nie można używać spacji)*
- *$i (nie można używać znaku $)*
- *if (if to słowo kluczowe)*

Aby kod był bardziej czytelny, przestrzegajmy poniższych (umownych) reguł:

   I. *nazwy zmiennych piszemy małymi literami: i, file*
   II. *nazwy stałych (zadeklarowanych przy pomocy #define) piszemy wielkimi literami: SIZE*
   III. *nazwy funkcji piszemy małymi literami: printf*
   IV. *wyrazy w nazwach oddzielamy jedną z konwencji:*
      - *oddzielanie podkreśleniem: open_file*
      - *konwencja wielbłądzia: openFile*
      - *konwencja pascalowska (ewentualnie): OpenFile*
6. Podstawowe typy zmiennych/stałych:
   I. ***char** - jednobajtowe liczby całkowite, służy do przechowywania znaków;*

II. ***int***- *typ całkowity, o długości domyślnej dla danej architektury komputera;*

III. ***float*** *- typ zmiennopozycyjny (zwany również zmiennoprzecinkowym), reprezentujący liczby rzeczywiste (4 bajty);*

IV. ***double*** *- typ zmiennopozycyjny podwójnej precyzji (8 bajtów);*

V. ***bool*** *(tylko C99) (wymaga dołączenia stdbool.h) - typ logiczny*

np. `int zmienna = 3;`

Więcej typów zmiennych/stałych na wykładzie.

7. Podstawowe funkcje I/O.

Funkcja **`printf()`** przyjmuje postać:

```
printf(format, argument1, argument2, ...);
```

```
- podstawowe wymagania
      -- dołączyć bibliotekę przy linkowaniu;
            #include <stdio.h>;
      -- argument funkcji dowolny ciąg znaków ujęty w cudzysłów "";
      -- tekst w jednej linii programu;
      -- w tekstach stosowane znaki niegraficzne:
            \n - nowa linia
            \t - tabulacja
            \b - cofanie
            \r - powrót karetki
            \f - nowa strona
      -- funkcja może mieć dowolną liczbę argumentów oddzielonych przecinkami;
      -- specyfikacja przekształcenia - umieszczony w cudzysłowach znak % i
                                    wszystko po nim aż do znaku przekształcenia;
            --- znaki przekształcenia:
                  d - argument przekształcany do postaci dziesiętnej;
                  f - argument traktowany jako liczba float, double
                      przekształcany do postaci mmm.dddddd;
                  e - argument traktowany jako liczba float, double
                      przekształcany do postaci m.ddddddE+xx;
                  u - argument przekształcany do postaci dziesiętnej bez
                      znaku;
                  c - argument traktowany jako pojedynczy znak;
                  s - argument traktowany jako ciąg znaków;
            --- znaki modyfikujące sposób wyprowadzania wyników
                   (nieobowiązkowe);
                  minus (-) - dosunięcie argumentu do początku pola;
                  plus (+) - wyprowadzenie znaku + przed liczbami dodatnimi;
                  ciąg_cyfr - określa minimalny rozmiar pola;
                  kropka (.) - oddziela rozmiar pola od dokładności;
                  ciąg_cyfr określający dokładność - liczba cyfr po kropce;
                  litera l - wskazuje na argument typu long (a nie int) lub
                             double (a nie float);
                  litera h - wskazuje na argument typu short (a nie int);
```

Przykładowe wywołanie:

```
int i = 500;
printf("Liczbami całkowitymi są na przykład %d oraz %d.\n", 1, i);
```

Znak \n oznacza przejście do nowej linii. Więcej znaków specjalnych na wykładzie. Po procencie następuje specyfikacja, jak wyświetlić dany argument. Pełna lista symboli

specyfikujących dany typ znajduje się w dokumentacji MSDN: https://msdn.microsoft.com/pl-pl/library/hf4y5e3w.aspx

Funkcja **scanf()** przyjmuje postać:

```
scanf(format, adres_zmiennej);
```

```
- podstawowe wymagania
      -- dołączyć bibliotekę przy linkowaniu;
            #include <stdio.h>;
      -- pozwala nadać wartości zmiennym - dane z klawiatury;
      -- argument pierwszy określa jak interpretować wprowadzone dane;
      -- argument drugi i kolejne określają której zmiennej przypisać wartość;
            --- każdą zmienną musi poprzedzać & (adres);
      -- argumenty zaczynając od drugiego muszą być adresami zmiennych;
      -- funkcja zwraca wartość typu int (liczba przeczytanych stałych);
```

Aby przekazać adres zmiennej, nazwę zmiennej poprzedzamy operatorem **&** (tzw. **etka, ampersand**). Przykład użycia:

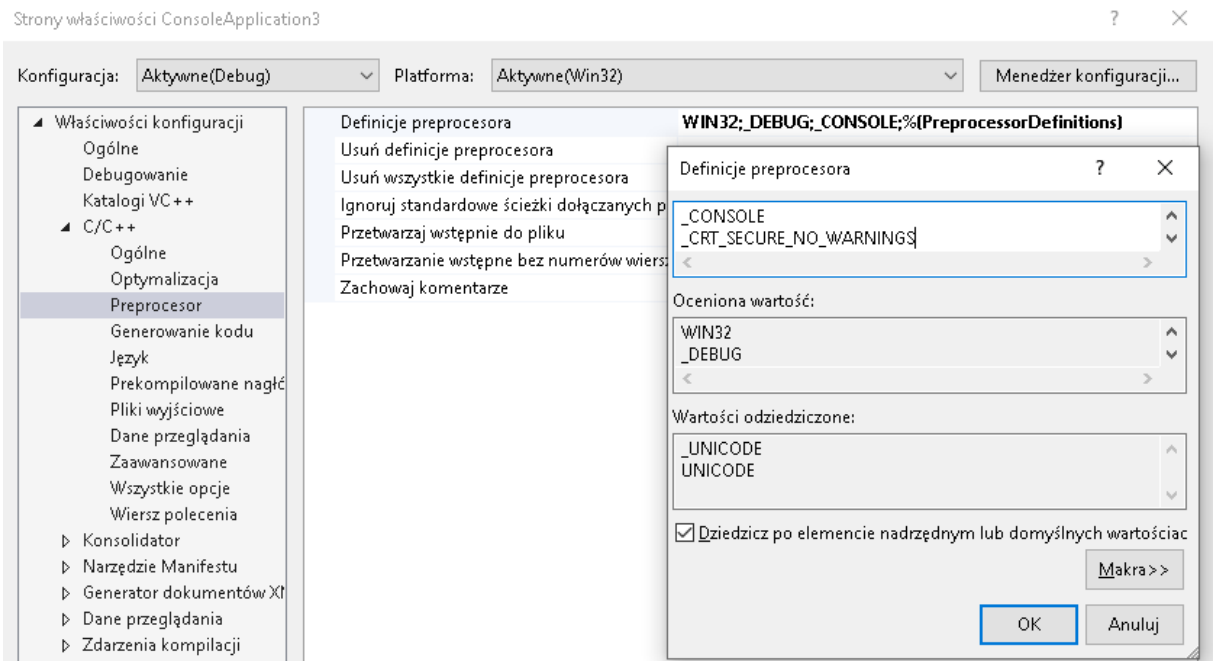
```
int liczba = 0;

printf ("Podaj liczbe: ");

scanf ("%d", &liczba);
```

UWAGA! Funkcja scanf() w środowisku VS może wymagać wyłączenia ostrzeżeń poprzez zastosowanie dyrektywy preprocesora: `#define _CRT_SECURE_NO_WARNINGS`

We właściwościach projektu dopisz w Definicje preprocesora: `_CRT_SECURE_NO_WARNINGS`

Lub w programie użyj dyrektywy:

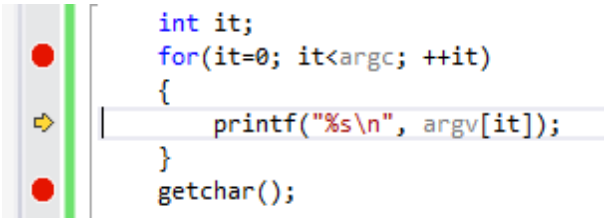
```
#pragma warning (disable: 4996)
```



8. Praca w trybie **Debug**. Domyślnie Twój projekt kompilowany jest w trybie **Debug** (definicja w I części konspektu).

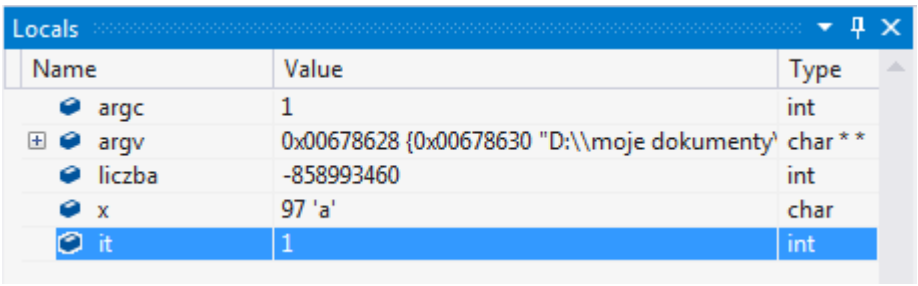
Dostępny jest również tryb **Release**. Jest to tryb dla wersji produkcyjnej (końcowej). Jego kompilacja trwa dłużej z powodu optymalizacji kodu. Praca krokowa w przypadku kompilacji **Release** może być niedostępna.

9. **Praca krokowa** w trybie **Debug**. F11 dokonuje szczegółowej pracy krokowej, odnosząc się do każdej pojedynczej instrukcji. Kombinacja F11 + Shift pozwala na wyjście z instrukcji zagnieżdżonych. Natomiast F10 nie zagłębia się w zewnętrzne funkcje. F5 przechodzi do kolejnego break point'a. **Break point**'y pozwalają na sprawną pracę krokową. Aby ustawić break point'a (czerwona kropka), należy nacisnąć kursorem na bocznej, pionowej belce, jak na poniższym przykładzie:



Następnie należy uruchomić nasz program: F5. Żółta strzałka określa naszą aktualną pozycję. Najeżdżając kursorem na zmienną możemy odczytać jej obecną wartość:

Te same informacje dla całego stosu zmiennych znajdziemy w zakładce **Lokalne** (**Locals**) (ewentualnie **Automatyczne** (**Autos)**) u dołu ekranu:



10. Wyczyść zawartość funkcji `main()`. Zadeklaruj dwie zmienne typu `int` (x i y). Nadaj dowolną wartość drugiej. Przypisz wartość y do x. Zwiększ x o 3, następnie (w kolejnej linijce) przemnóż przez 2. Potem wypisz obie na ekran uzupełniając tekstem:
    Początkowa wartość:
    Końcowa wartość:
    W trybie pracy krokowej przeanalizuj proces zmiany wartości. Jaką wartość miały zmienne przed nadaniem pierwszych wartości?
11. Dyrektywy preprocesora (definicja w I części konspektu). Do najważniejszych dyrektyw preprocesora języka C zaliczamy:
    I. ***#include*** *… - dyrektywa włączająca tekst innego pliku źródłowego w miejscu jej wystąpienia w pliku podlegającym aktualnie przetwarzaniu, przy czym możliwe jest zagłębione występowanie dyrektywy include,*

II. ***#define*** *… - definiuje stałe i makroinstrukcje (pseudofunkcje)*
III. ***#undef*** *… - usuwa definicje stałej lub makra*
IV. ***#if*** *… - dyrektywy kompilacji warunkowej*
V. ***#elif*** *… - działa podobnie jak else if w języku C*
VI. ***#endif*** *… - oznacza koniec bloku kompilacji warunkowej*
VII. ***#ifdef*** *… - znaczy to samo co #if defined(…)*
VIII. ***#ifndef*** *… - znaczy to samo co #if !defined(…)*

Predefiniowane makra (wybrane) – nazwy zaczynają się od podwójnych podkreśleń:

__DATE__ - data w momencie kompilacji (%s)
__TIME__ - godzina w momencie kompilacji (%s)
__FILE__ - nazwa pliku, który aktualnie jest kompilowany przez kompilator (%s)
__LINE__ - definiuje bieżący numer wiersza (%d)

Korzystając z dyrektywy **#define** zadeklaruj stałą Y tą samą wartością, co zmienną y w zad. 10. Zadeklaruj również wyrażenie w postaci ADD(a, b) a+b, którym to zastąp dodawanie w programie. Sprawdź poprawność działania aplikacji. Następnie zdefiniuj stałą VER, jeżeli stała ta będzie równa 2 ma wykonywać się kod jak w zadaniu 11, w przeciwnym wypadku jak w zadaniu 10 (bez wykorzystania stałych i makra). Wykorzystaj makra __DATE__ oraz __TIME__ .

12. Utwórz nowy projekt i napisz program, który będzie wczytywał rok urodzenia, na podstawie którego obliczy wiek i wypisze go na ekran.
13. Przeanalizuj w trybie pracy krokowej następujące przykłady do wykładów:
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/WW1.pdf
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/WW1_1.pdf
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/WW2.pdf

*Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.