# Wstęp do Informatyki i Programowania
# Laboratorium: Lista 0
# Środowisko programowania

Przemysław Kobylański

## Wprowadzenie

Każdy program w C musi zawierać przynajmniej funkcję o nazwie `main()`:

```
int main()
{
  ...
  return 0;
}
```

Aby możliwe było czytanie danych i drukowanie wyników, trzeba dołączyć plik nagłówkowy biblioteki standardowego wejścia i wyjścia (stdio). Wykonuje się to umieszczając na początku programu dyrektywę `include`:

```
#include <stdio.h>
int main()
{
  ...
  return 0;
}
```

Każda zmienna jaką używamy w funkcji do przechowywania wartości musi być zadeklarowana na początku funkcji:

```
#include <stdio.h>
int main()
{
  int x;
  int y;
  int z;
  ...
  return 0;
}
```

Do przeczytania wartości służy funkcja `scanf()`:

```
#include <stdio.h>
int main()
{
  int x;
  int y;
  int z;
  scanf("%d", &x);
  scanf("%d", &y);
  ...
  return 0;
}
```

Wartości całkowite typu `int` czyta się formatem `%d`, natomiast wartości rzeczywiste typu `float` (pojedynczej dokładności) formatem `%f`.

Aby podstawić wyliczoną wartość pod zmienną należy użyć instrukcji podstawienia:

```
#include <stdio.h>
int main()
{
  int x;
  int y;
  int z;
  scanf("%d", &x);
  scanf("%d", &y);
  z = x+y;
  ...
  return 0;
}
```

Wartość dowolnego wyrażenia (w tym tak prostego jak złożonego z pojedynczej zmiennej) można wydrukować za pomocą funkcji `printf()`:

```
#include <stdio.h>
int main()
{
  int x;
  int y;
  int z;
  scanf("%d", &x);
  scanf("%d", &y);
  z = x+y;
  printf("Suma␣%d␣i␣%d␣wynosi␣%d.\n", x, y, z);
  return 0;
}
```

Wartości całkowite drukuje się przy użyciu formatu `%d`, natomiast rzeczywiste formatem `%f`.

Opis podstawowych poleceń systemu Linux, w tym korzystanie z prostego edytora tekstowego, znajduje się w dodatku A.

Po utworzeniu pliku ze źródłami programu, należy go skompilować za pomocą polecenia `clang` opisanego w dodatku B.

Aby ułatwić kompilację i usuwanie niepotrzebnych plików będziemy przygotowywać pliki `Makefile` dla polecenia `make`. Krótki opis tego polecenia zamieszczono w dodatku C.

## Zadanie 1

Napisz program, który wczytuje dwie liczby rzeczywiste (typu `float`) i wyprowadza ich sumę, różnicę, iloczyn i iloraz.

## Zadanie 2

Napisz program, który przekształca temperaturę podaną w stopniach Celsjusza na temperaturę w stopniach Fahrenheita. Związek pomiędzy temperaturą wyrażoną w stopniach Celsjusza i stopniach Fahrenheita wyraża się wzorem $F = 1.8 \cdot C + 32$.

# A Podstawowe polecenia Linuxa

| **polecenie** | **opis** | **przykłady** |
|---|---|---|
| `mkdir NAZWA`<br>`mkdir ŚCIEŻKA/NAZWA` | stworzenie katalogu | `mkdir zadanie1`<br>`mkdir zadanie1/src` |
| `rmdir NAZWA`<br>`rmdir ŚCIEŻKA/NAZWA` | usunięcie katalogu<br>(katalog musi być pusty) | `rmdir zadanie1`<br>`rmdir ../zadanie2/src` |
| `cd ŚCIEŻKA` | przejście do katalogu | `cd zadanie1/src`<br>`cd ../../zadanie2/src` |
| `ls`<br>`ls -l`<br><br>`ls ŚCIEŻKA`<br>`ls -l ŚCIEŻKA` | wyświetlenie zawartości<br>bieżącego katalogu<br>(opcja pełnego opisu plików);<br>wyświetlenie zawartości<br>katalogu | `ls`<br>`ls -l`<br><br>`ls zadanie1/src`<br>`ls -l ../../zadanie2/src` |
| `nano`<br>`nano ŚCIEŻKA/NAZWA` | edycja pliku tekstowego<br>(na dole ekranu opis<br>skrótów klawiszowych dla<br>podstawowych operacji,<br>np. ^X oznacza Ctrl-x) | `nano`<br>`nano src/main.c` |
| `man POLECENIE` | opis polecenia | `man man`<br>`man nano` |

# B Skrócony opis polecenia `clang`

Do kompilacji programów w języku C używać będziemy polecenia `clang`.

Załóżmy, że następujące źródła programu znajdują się w pliku `witaj.c`:

```
#include <stdio.h>
int main()
{
  printf("witaj swiecie!\n");
}
```

Aby skompilować program wystarczy wydać polecenie: `clang witaj.c`

Skompilowany program wykonywalny znalazł się w pliku `a.out`

Można teraz go uruchomić wydając polecenie: `./a.out`

Zwróć uwagę na kropkę w powyższym poleceniu, która wskazuje, że plik `a.out` znajduje się w bieżącym katalogu.

Jeśli chcemy aby wykonywalny program znalazł się w pliku `witaj`, zamiast `a.out`, należy wydać polecenie: `clang -o witaj witaj.c`

Opcja `-o` służy do określenia nazwy pliku wykonywalnego.

Programy kompilować będziemy z dodatkową opcją `-Wall` nakazującą kompilatorowi informować o wszystkich ostrzeżeniach:

```
$ clang -Wall -o witaj witaj.c
witaj.c: In function ‘main’:
witaj.c:5: warning: control reaches end of non-void function
$
```

Ostrzeżenie **control reaches end of non-void function** oznacza, że w funkcji main (dokładniej w wierszu 5 pliku `witaj.c`) sterowanie, tj. wykonywanie, osiąga klamrę zamykającą definicję funkcji, mimo tego, że została ona zadeklarowana jako zwracająca wartość typu całkowitego (`int`).

Na program należy poprawić dopisując między instrukcją `printf(...)` a klamrą `}` instrukcję `return 0;`

Program po poprawieniu wygląda następująco:

```
#include <stdio.h>
int main()
{
  printf("witaj swiecie!\n");
  return 0;
}
```

Tym razem kompiluje się beż żadnych ostrzeżeń:

```
$ clang -Wall -o witaj witaj.c
$
```

Jeśli w programie korzysta się z funkcji bibliotecznych, to należy za pomocą opcji `-l` podać nazwę wykorzystywanej biblioteki.

Dla przykładu, jeśli program `calkowanie.c` korzysta z funkcji matematycznych, to kompilujemy go poleceniem: `clang -o calkowanie calkowanie.c -lm`

Opcja `-lm` w powyższym poleceniu zawiera nazwę `m` biblioteki funkcji matematycznych.

Więcej o poleceniu `clang` można dowiedzieć się z jego opisu uzyskanego poleceniem: `man clang`

Opis ten zawiera ponad 8800 wierszy i większość opisanych w nim opcji wykracza poza zakres wykładu ze Wstępu do Informatyki i Programowania a nawet innych, bardziej zaawansowanych, kursów.

# C Skrócony opis polecenia `make`

Jeśli często wykonujemy kompilację, a szczególnie w przypadku dużych projektów podzielonych na wiele plików źródłowych, przydatne jest użycie polecenia `make`.

Polecenie to czyta plik `Makefile`, w którym za pomocą reguł opisuje się w jaki sposób jakie pliki otrzymać z jakich plików:

```
CO: Z CZEGO
        JAK
```

Ważne jest aby polecenia w opisie JAK zapisane były po znaku tabulacji.

Powróćmy do przykładowego programu `witaj.c` z dodatku B.

W pliku `Makefile` wpiszemy następujące trzy reguły:

```
all: witaj

witaj: witaj.c
        clang -Wall -o witaj witaj.c

clean:
        rm -f witaj *~
```

Pierwsza reguła:

```
all: witaj
```

opisuje jak wszystko (ang. all) skompilować. Podano w niej, że w tym celu potrzebny będzie plik `witaj`. Część JAK reguły jest pusta, gdyż jeśli otrzyma się plik `witaj`, to nic już nie trzeba robić.

Druga reguła:

```
witaj: witaj.c
        clang -Wall -o witaj witaj.c
```

opisuje, że plik `witaj` otrzyma się z pliku `witaj.c` wykonując polecenie: `clang -Wall -o witaj witaj.c`

Ostatnia reguła:

```
clean:
        rm -f witaj *~
```

opisuje co zrobić by posprzątać (usunąć niepotrzebne pliki). W tym celu należy wykonać polecenie: `rm -f witaj *`

Koniecznie zwróć uwagę na znak tyldy `~`. Bez niego usunięte byłyby wszystkie pliki (wzorzec `*` pasuje do każdej nazwy pliku) a tak zostaną usunięte tylko stare (poprzednie) wersje plików (w systemach Linux/Unix poprzednia wersję pliku oznacza się dopisując na końcu jego nazwy tyldę).

Opcja `-f` w poleceniu `rm` wskazuje, że kasowanie plików powinno odbywać się bez każdorazowego pytania o potwierdzenie i jakichkolwiek ostrzeżeń jeśli nie ma plików do skasowania o zadanej nazwie.

Wykonajmy czyszczenie. W tym celu uruchamiamy polecenie: `make clean`

Po jego wykonaniu na pewno w bieżącym katalogu nie ma pliku wykonywalnego `witaj`.

Aby go otrzymać wykonamy polecenie `make all` albo krócej `make`

Jeśli jeszcze raz spróbujemy skompilować plik `witaj.c` wydając polecenie `make` otrzymamy komunikat **make: Nothing to be done for 'all'** co oznacza, że nic nie ma do roboty bo plik źródłowy `witaj.c` jest starszy niż plik wykonywalny `witaj`.

Kontrola czasu ostatniej modyfikacji plików źródłowych oraz wynikowych, wykonywana przez polecenie `make`, umożliwia w dużych projektach (wiele plików źródłowych) kompilację tylko tych fragmentów programu (tych plików źródłowych), które uległy zmianie od ostatniej ich kompilacji.

Więcej o poleceniu `make` dowiesz się na kursie Środowisko Programisty a wcześniej czytając podręcznik *GNU Make. A Program for Directing Recompilation* (znajdziesz go na stronach `http://www.gnu.org/software/make/` )