

Wstęp do Informatyki i Programowania

Laboratorium: Lista 0

Środowisko programowania

Przemysław Kobyłański

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 podstawienia¹:

```

#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;
}

```

¹Prawdę mówiąc w języku C nie ma instrukcji podstawienia ale będzie o tym więcej powiedziane na wykładzie.

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.

Polecenie 1

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

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

B Skrócony opis polecenia clang

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

Założmy, 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`).

Nasz 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 bez 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` wpisujemy 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 *~`

Konieczne 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 Recompile* (znajdziesz go na stronach <http://www.gnu.org/software/make/>)