Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki
Katedra Elektrotechniki Przemysłowej i Automatyki
Zakład Urządzeń i Systemów Automatyki

Programowanie komputerów 2

Wprowadzenie do języka C

Instrukcja laboratoryjna
(wersja robocza)

Paweł Strączyński
2017

1. Struktura programu w języku C, podstawowe dyrektywy preprocesora, komentarze.

Szkielet najprostszego programu w języku C przedstawiono na listingu 1.1. Programy w języku C składają się z funkcji i zmiennych. Funkcje zawierają zbiór wykonywanych operacji (instrukcji). Zmienne przechowują przetwarzane w wyniku działania programu wartości. Przedstawiony poniżej program składa się z jednej funkcji – `main()`. Program w języku C może zawierać dowolną ilość funkcji, jednak zawsze musi istnieć funkcja `main()` - jest to funkcja główna od której program zaczyna działanie. Jak widać język C umożliwia dwa sposoby tworzenia komentarzy w kodzie. Komentarzem jest każdy fragment kodu zawarty między znakami `/*` oraz `*/`. Komentarz taki może zawierać więcej niż jedna linię. Drugi rodzaj tworzenia komentarzy rozpoczyna się od znaków `//` i kończy wraz z końcem linii.

Listing 1.1. Szkielet programu w języku C

```
/* DYREKTYWY PREPROCESORA

   ZMIENNE GLOBALNE */

void main()
{
      // ZMIENNE LOKALNE
      //INSTRUKCJE
}
```

Zwykle program w języku C zaczyna się od dyrektyw preprocesora `#include` oraz `#define`.

Dyrektywa `#include` dołącza do programu zawartość innego pliku. Służy zwykle do zaimportowania zawartości biblioteki (zbioru funkcji) wykorzystywanej w programie. Poniżej przedstawiono sposób użycia dyrektywy `#include` :

```
#include <plik_do_zalaczenia>

#include "plik_do_zalaczenia"
```

Jeżeli nazwa pliku do załączenia przy dyrektywie `#include` ujęta będzie w nawiasy ostre to oznacza to że kompilator „poszuka" go wśród własnych plików nagłówkowych. Jeżeli nazwa będzie ujęta w cudzysłów to kompilator poszuka pliku w katalogu w którym znajduję się kompilowany plik. Dopuszczalne jest także wskazanie położenia pliku z wykorzystaniem ścieżki względnej lub bezwzględnej.

Dyrektywa `#define` służy do definiowania stałych oraz markoinstrukcji.

Przykładowe użycie dyrektyw `#include` oraz `#define` przedstawiono na listingu 1.2.

Listing 1.2. Przykładowy program w języku C wykorzystujący dyrektywy preprocesora

```
#include <stdio.h>
#define PI 3.14

int main()
{
      printf("%f",PI);
      getchar();

      return 0;
}
```

Program przedstawiony powyżej nie robi nic innego jak tylko wyświetla zawartość zdefiniowanej stałej `PI`. Wykorzystano tu funkcje `printf()` z zaimportowanej poleceniem `#include <stdio.h>` biblioteki standardowej. Funkcja `getchar()` której użyto zatrzymuje działanie programu – oczekuje na wciśniecie klawisza. Użycie słowa kluczowego `return` oznacza iż funkcja główna `main()` w wyniku swojego działania zwraca wartość – w tym przypadku 0.

2. Zmienne – typy, deklaracje

Zmienna to fragment pamięci, posiadający nazwę i służący do przechowywania wartości (danych) wykorzystywanych podczas działania programu. Każda zmienna w języku C posiada typ określający zakres wartości jakie może przyjmować. Podstawowe typy zmiennych w języku C przedstawiono w tabeli 2.1.

Tabela 2.1. Podstawowe typy zmiennych w języku C

Typ	**Rozmiar (w bajtach)**	**Przeznaczenie**
char	1	jednobajtowe liczby całkowite, znaki
short int	2	liczba całkowita
int	4	liczba całkowita
long int	8	liczba całkowita
float	4	liczby zmiennoprzecinkowe
double	8	liczby zmiennoprzecinkowe
void	-	brak wartości

Domyślnie typ znakowy oraz typy całkowitoliczbowe przechowują liczby ze znakiem. Aby przekształcić je na liczby bez znaku należy przed nazwą typu dodać specyfikator `unsigned`. Deklarowanie zmiennych w języku C odbywa się według schematu jak poniżej:

```
typ zmienna;
```

Przykłady deklaracji zmiennych przedstawiono na listingu 2.1. Zmienne które są zadeklarowane poza funkcją są zmiennymi globalnymi. Zmienne deklarowane wewnątrz funkcji to zmienne lokalne.

Listing 2.1. Przykład deklaracji funkcji w języku C

```
int foo, bar;
char foobar= "A";
float = 3.14;
```

Jak widać powyżej podczas deklaracji zmiennej można dokonać przypisania jej wartości. Operatorem przypisania w języku C jest znak `=`. Nazwa zmiennej może składać się z liter i cyfr, jednak zawsze musi zaczynać się literą. W nazwie zmiennej nie mogą występować spacje. Umownie przyjmuje się, że nazwy zmiennych pisane są małymi literami, a nazwy stałych wielkimi. Nazwą zmiennej nie może być jedna z zastrzeżonych słów kluczowych języka C. Słowa zastrzeżone przedstawiono w tabeli 2.2.

Tabela 2.1. Zastrzeżone słowa kluczowe języka C

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

3. Podstawowe operatory

W poprzednim punkcie wspomniano, że operatorem przypisania w języku C jest znak `=`. Operator ten może być użyty kaskadowo – obliczanie przypisań następuje od prawej do lewej strony.

W tabeli 3.1 przedstawiono operatory arytmetyczne w języku C. Dwuargumentowe operatory arytmetyczne mogą być również łączone z operatorem przypisania w celu uzyskania skróconego zapisu – listing 3.1.

Tabela 3.1. Operatory arytmetyczne w języku C

Operator	Opis
++	inkrementalna
--	dekrementacja
+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
%	modulo – reszta z dzielenia

Dwa pierwsze operatory przedstawione w tabeli (inkrementalna i dekrementacja) to operatory jednoargumentowe. Mogą być stosowane zarówno przed (pre-) jak i po (post-) argumencie do którego się odnoszą. Różnice w użyciu przedstawino na listingu 3.1. Pozostałe operatory wymagają dwóch argumentów i realizują podstawowe operacje arytmetyczne.

Listing 3.1. Przykład wykorzystania operatorów arytmetycznych

```
int foo, bar, foobar, quux, quuz;
foo = bar = 1; //kaskadowe przypisanie wartości
foo = foo + 1; //inkrementacja zmiennej z użyciem operatora dodawania
foo += 1; //inkrementacja zmiennej - zapis skrócony
foo++; //inkrementacja zmiennej - z użyciem operatora inkrementacji
quux = foo++; //postinkrementacja - zmienna quux=1, zmienna foo=2
quuz = ++foo; //preinkrementacja - zmienna quux=2, zmienna foo=2
foobar = foo % bar;
```

W tabelach 3.2, 3.3 oraz 3.4 przedstawiono odpowiednio operatory: porównania, logiczne oraz bitowe występujące w języku C.

Tabela 3.2 Operatory porównania

Operator	Opis
==	równe
!=	różne
<	mniejsze
>	większe
<=	mniejsze lub równe
>=	większe lub równe

Tabela 3.3 Operatory logiczne

Operator	Opis
||	suma logicza (or)
&&	iloczyn logiczny (and)
!	negacja (not)

Tabela 3.4 Operatory bitowe

Operator	Opis
|	suma bitowa (or)
&	iloczyn bitowy (and)
^	alternatywa wykluczająca (xor)
~	negacja bitowa (not)
>>	przesunięcie bitowe w prawo
<<	przesunięcie bitowe w lewo

Operator przypisania może być być również łączony z jednym z następujących operatorów bitowych: `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `<<`, `>>`.

4. Instrukcja warunkowe

W języku C do warunkowego wykonywania danego fragmentu programu służą:

- instrukcja warunkowa `if` (`if-else`)
- instrukcja wyboru wielokrotnego `switch`
- operator trójargumentowy `?`

Najczęściej wykorzystywaną instrukcją warunkową jest instrukcja `if`. Przyjmuje ona następującą konstrukcję:

```
if (wyrażenie_warunkowe)
{
      /*instrukcje wykonywane jeśli spełniony warunek*/
}
else
{
      /*instrukcje wykonywane jeśli warunek nie spełniony*/
}
```

Przy czym użycie słowa kluczowe `else` jest opcjonalne.

Przykłady wykorzystania instrukcji warunkowej `if` przedstawiono na listingu poniżej.

Listing 4.1. Przykładowe wyrażenia warunkowe w języku C

```
int foo=2, bar = 3;
if (foo == 2)
{
      printf("ROWNE");
}
else
{
      printf("ROZNE");
}

if (( foo == 2) && (bar !=2)) foo++; //Łączenie warunków
```

Zamiast konstrukcji `if-else` można użyć skróconego zapisu z wykorzystaniem operatora `?`. Wyrażenie to przyjmuje następująca konstrukcję:

```
wyrażenie_logiczne ? wart_prawda : wart_falsz
```

Przykład wykorzystania operatora trójargumentowego `?` przedstawiono poniżej.

Listing 4.2. Przykładowe wyrażenia warunkowe wykorzystujące operator ?

```
int foo=3; bar=7;
min = (foo < bar) ? foo : bar;
max = (foo > bar) ? foo : bar;
```

Do wykonywania instrukcji warunkowych wielokrotnego wyboru, gdzie sprawdza się czy wartość wyrażenia pasuje do jednej z kilku wartości służy instrukcja `switch`. Instrukcja ta przyjmuje następująca postać:

```
switch(wyrażenie)
{
      case wart_1:
            /*instrukcje*/
            break;
      case wart_2:
            /*instrukcje*/
            break;
      case wart_n:
            /*instrukcje*/
            break;
      default:
            /*instrukcje*/
}
```

Jeżeli wyrażenie jest równej której z wartości stojących przy słowie kluczowym `case` oznacza to, że wykonywane będą instrukcje z tego wariantu do chwili napotkania słowa kluczowego `brake`. Gdyby słowa tego zabrakło, to wykonywane były by dalsze warianty aż do napotkania tego słowa albo zakończenia instrukcji warunkowej. Słowo kluczowe `default` pełni tu rolę odpowiednika słowa `else` w instrukcji `if` – instrukcje te będą wykonywane jeżeli żaden z wariantów nie jest równy wyrażeniu.

Listing 4.3. Przykładowe użycie instrukcji warunkowej switch

```
int foo=2;
switch(foo)
{
      case 1:
            printf("Zmienna foo jest równa jeden";
            break;
      case 2:
            printf("Zmienna foo jest równa dwa";
            break;
      case 3:
            printf("Zmienna foo jest równa trzy";
            break;
      case 4:
            printf("Zmienna foo jest równa cztery";
            break;
}
```

5. Pętle

Pętle wykorzystuje się wszędzie tam gdzie zachodzi potrzeba wykonywania wielokrotnie tego samego ciągu instrukcji.

Do wykonywania tego samego ciągu instrukcji dopóki spełniony jest dany warunek w języku C służy pętla `while`. Przyjmuje ona następującą konstrukcję:

```
while(wyrażenie)
{
      /*instrukcje do wykonania w pętli*/
}
/*pozostałe instrukcje*/
```

Problemem z pętlą `while` może okazać się fakt, że może zdarzyć się przypadek gdy nie wykona się ona ani razu – wyrażenie nie jest spełnione. Aby mieć pewność że pętla wykonała się co najmniej raz należy użyć pętli `do while`. Zasadniczą różnica między pętlą `while` jest to, że sprawdza ona warunek pod koniec każdego obiegu pętli. Pętle można stworzyć według schematu:

```
do
{
      /*instrukcje do wykonania w pętli*/
}while(warunek);
/*pozostałe instrukcje*/
```

Do wykonywania obiegu ustaloną ilość razy często wygodniejsza w użyciu jest pętla `for`. Ogólna postać pętli `for` wygląda następująco:

```
for(wyrażenie1; wyrażenie2; wyrażenie3)
{
      /*instrukcje do wykonania w pętli*/
}
/*pozostałe instrukcje*/
```

gdzie:

- wyrażenie1 – instrukcja wykonywana przed pierwszym obiegiem pętli -zwykle służy do inicjalizacji licznika obiegu pętli.
- wyrażenie2 – warunek zakończenia pętli - zwykle sprawdzenie czy licznik obiegu pętli osiągnął żądaną wartość;
- wyrażenie3 – instrukcja wykonywana po każdym obiegu pętli -zwykle służy do inkrementacji iterowanej zmiennej.

Przykłady użycia przedstawionych pętli zaprezentowano na listingu 5.1

Listing 5.1. Przykładowe użycia pętli

```
int foo = 1;
while(1)
{
      /*pętla nieskończona*/
}

for(;;;)
{
      /*pętla nieskończona*/
}

while(foo < 5)
{
      printf("%d",foo);
      foo+=1;
}

for(foo = 1; foo < 5; foo++)
{
      printf("%d",foo);
}
```

6. Operacje we/wy – interakcja z użytkownikiem

W poprzednich punktach używana była instrukcja `printf()` umożliwiająca wyświetlanie użytkownikowi na ekranie komputera (standardowym wyjściu) pewnych informacji. Jej deklaracje przedstawiono na listingu 6.1.

Listing 6.1. Deklaracja funkcji printf

```
int printf(const char *tekst, argument1, argument2,...);
```

Funkcja ta wyświetla na ekranie sformatowany tekst zgodnie z podanym formatem. Gdy w łańcuchu znakowym (tekście) do wyświetlenia wystąpi specyfikator formatu zaczynający się od znaku `%` wówczas w jego miejsce zostanie wstawiona wartość argumentu. Funkcja powinna pobierać tyle dodatkowych argumentów ile specyfikatorów formatu zostało użyte w łańcuchu znakowym. Specyfikator formatu buduję się według następującego schematu:

```
specyfikator = %[znacznik][szerokość][.precyzja][typ]
```

gdzie:

- `[znacznik]` – określa dodatkowy znacznik poprzedzający liczbę np. 0 spowoduje uzupełnianie pustych miejsc zerami,
- `[szerokość]` – określa minimalną szerokość wynikowego tekstu wstawionego w miejscu użycia specyfikatora – pole domyślnie uzupełniane jest spacjami z lewej strony,
- `[.precyzja]` – liczba cyfr wyświetlanych po kropce,
- `[typ]` -określa typ argumentu (tabela 6.1)

Tabela 6.1 Typy argumentów używane w formatowaniu łańcuchów znakowych

Typ	**Opis**
d,i	liczba całkowita dziesiętna
u	liczba całkowita dziesiętna, bez znaku
x,X	liczba szesnastkowa, bez znaku
o	liczba ósemkowa, bez znaku
f	liczba rzeczywista
e, E	liczba rzeczywista format naukowy
s	łańcuch znakowy
c	znak

Listing 6.2. Przykład użycia funkcji printf

```
int foo = 1;
float bar = 14.32432
char foobar = "A";
printf("Zmienna foo = %d\nZmienna bar = %f\nZmienna spam = %c",foo,bar,foobar);
```

Do wprowadzania danych przez użytkownika użyć można funkcji `scanf()`. Deklaracje funkcji przedstawiono na listingu 6.3.

Listing 6.3. Deklaracja funkcji printf

```
int scanf(const char *tekst,...);
```

Przykładowy program używający funkcji `scanf()` przedstawiono poniżej. Należy zauważyć, że jako argument podawany jest adres zmiennej - operator `&` przed nazwą zmiennej.

Listing 6.2. Przykładowy program z użyciem funkcji scanf

```
#include <stdio.h>

int main()
{
	int foo, bar;
	printf("Podaj 1 liczbę:");
	scanf("%d",&foo);
	printf("Podaj 2 liczbę:");
	scanf("%d",&bar);
	printf("%d x %d = %d",foo,bar,foo*bar);
	return 0;
}
```

Tabela 6.2 Znaki specjalne używane w ciągach znakowych

Typ	**Opis**
\n	nowy wiersz
\t	tabulator poziomy
\v	tabulator pionowy
\a	alarm
\\	\ (backslash)
\’	apostrof
\”	cudzysłów
\?	znak zapytania

Zadania do wykonania

Zadanie 1

Napisać program który wyświetli na ekranie tekst jak poniżej.

```
**********************************************
**                                          **
**                HELLO WORLD               **
**                                          **
**********************************************
```

Zadanie 2

Napisać program który sprawdzi czy podana przez użytkownika liczba jest parzysta.

Zadanie 3

Napisać program który wyznaczy wartość liczby π według zależności poniżej:

$$\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = \frac{\pi}{4}$$

Dokładność obliczenia (liczba kroków iteracji) powinna być podawana przez użytkownika.

Zadanie 4

Napisać program wyznaczający średnią arytmetyczną liczb wprowadzonych przez użytkownika. Ilość liczb powinna być wprowadzana z klawiatury przez użytkownika.

Zadanie 5

Rozbudować program z zadania 4 tak, aby dla wprowadzonych liczb wyznaczał:

- wartość maksymalną,
- wartość minimalną.

Zadanie 6

Napisać program który dla podanej przez użytkownika liczby dziesiętnej wyświetli jej reprezentacje w systemie ósemkowym i szesnastkowym.