



Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 2

Kod przedmiotu: **ES1E3017**
(studia stacjonarne)

JĘZYK C - FUNKCJE

Numer ćwiczenia

INF25

Autor:
dr inż. Jarosław Forenc

Białystok 2017

Spis treści

1. Opis stanowiska	3
1.1. Stosowana aparatura	3
1.2. Oprogramowanie	3
2. Wiadomości teoretyczne.....	3
2.1. Struktura programu w języku C	3
2.2. Ogólna struktura funkcji w języku C.....	4
2.3. Umieszczanie definicji funkcji w programie.....	6
2.4. Klasyfikacja funkcji	8
3. Przebieg ćwiczenia.....	13
4. Literatura.....	16
5. Pytania kontrolne	16
6. Wymagania BHP.....	17

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

© Wydział Elektryczny, Politechnika Białostocka, 2017 (wersja 3.2)

Wszelkie prawa zastrzeżone. Żadna część tej publikacji nie może być kopiowana i odtwarzana w jakiegokolwiek formie i przy użyciu jakichkolwiek środków bez zgody posiadacza praw autorskich.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows (XP/ 7/10).

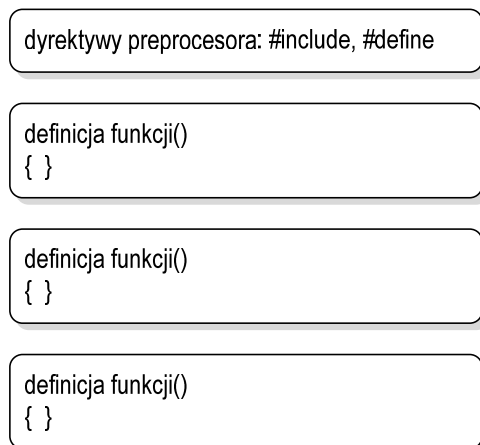
1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Microsoft Visual Studio 2008 Standard Edition lub Microsoft Visual Studio 2008 Express Edition zawierające kompilator Microsoft Visual C++ 2008.

2. Wiadomości teoretyczne

2.1. Struktura programu w języku C

W strukturze programu w języku C można wyróżnić dyrektywy preprocesora (np. `#include`, `#define`) oraz definicje funkcji (Rys. 1).

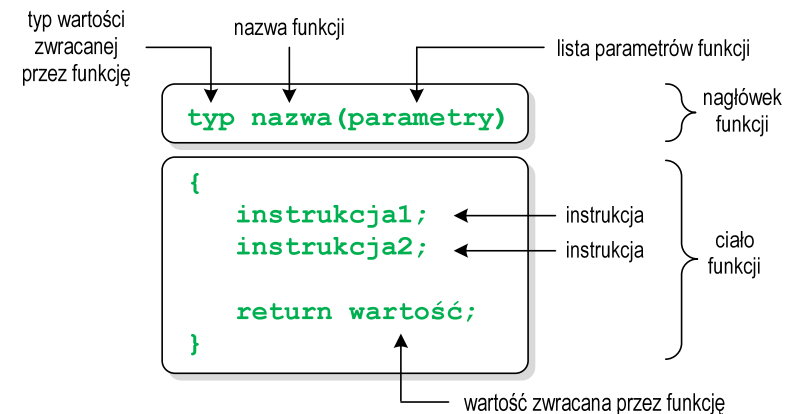


Rys. 1. Struktura programu w języku C

Każdy program musi zawierać przynajmniej jedną definicję funkcji. Funkcja ta powinna nazywać się `main()`. Od niej rozpoczyna się wykonanie całego programu. Oprócz funkcji `main()` w programie mogą występować także inne funkcje zdefiniowane przez użytkownika.

2.2. Ogólna struktura funkcji w języku C

Ogólną strukturę funkcji w języku C przedstawia Rys. 2. Funkcja składa się z **nagłówka** oraz **ciała** funkcji, które razem tworzą **definicję funkcji**. Nagłówek zawiera typ wartości zwracanej przez funkcję, nazwę funkcji oraz listę parametrów przekazywanych do niej podczas wywołania (uruchomienia). Ciało funkcji ograniczone jest nawiasami klamrowymi `{ }`. Pomiedzy nawiasami umieszczone są instrukcje. Funkcja kończy swoje działanie po wykonaniu instrukcji zawierającej słowo kluczowe `return`. Po `return` umieszcza się wartość, która zostanie zwrócona do miejsca wywołania funkcji. Typ tej wartości powinien być zgodny z nazwą typu umieszczoną w nagłówku funkcji. Słowo `return` może pojawić się wielokrotnie w ciele funkcji.



Rys. 2. Ogólna struktura funkcji w języku C

Poniżej zamieszczono przykład programu zawierającego dwie funkcje:

- `main()` - główna funkcja programu;
- `suma()` - funkcja obliczająca sumę dwóch liczb rzeczywistych.

Program zawierający funkcję obliczającą sumę dwóch liczb rzeczywistych.

```
#include <stdio.h>

float suma(float a, float b)
{
    float y;
    y = a + b;
    return y;
}

int main(void)
{
    float x1 = 10.0, x2 = 20.0, wynik;

    wynik = suma(x1,x2);
    printf("Wynik = %f\n", wynik);

    return 0;
}
```

Wynik uruchomienia programu:

```
Wynik = 30.000000
```

Wykonanie programu rozpoczyna się od funkcji **main()**. Gdy dochodzimy do instrukcji zawierającej funkcję **suma()**, to wywołanie tej funkcji powoduje przekazanie sterowania do jej pierwszej instrukcji. Do funkcji **suma()** przekazywane są dwa argumenty **x1** i **x2** typu **float**. Pierwszy parametr (a) otrzymuje wartość pierwszego argumentu wywołania funkcji (**x1**), natomiast drugi parametr (b) - wartość drugiego argumentu wywołania funkcji (**x2**). Powrót z funkcji (do miejsca zaraz po jej wywołaniu) następuje na skutek wykonania instrukcji **return**. Wartość zwracana przez funkcję podstawiana jest pod zmienną **wynik**.

Po słowie **return** może występować dowolne wyrażenie. Wyrażenie to często umieszczane jest w nawiasach, ale nie jest to konieczne. Funkcję **suma()** można zapisać w prostszy sposób:

```
float suma(float a, float b)
{
    return (a+b);
}
```

W wywołaniu funkcji jako argumenty mogą występować stałe liczbowe, nazwy zmiennych, wyrażenia arytmetyczne lub wywołania innych funkcji.

```
wynik = suma(10,20);
wynik = suma(x1,x2);
wynik = suma(x1*20+4,x1/x2);
wynik = suma(sin(x1),x1+x2);
printf("Wynik = %f\n", suma(x1,x2));
```

2.3. Umieszczanie definicji funkcji w programie

W programie przedstawionym w poprzednim rozdziale definicja funkcji **suma()** była umieszczona przed definicją funkcji **main()** (Rys. 3a).

(a)

dyrektywy preprocesora

definicja funkcji suma()
{ }

definicja funkcji main()
{ }

(b)

dyrektywy preprocesora

prototyp funkcji suma()
{ }

definicja funkcji main()
{ }

definicja funkcji suma()
{ }

Rys. 3. Kolejność funkcji w programie: (a) - funkcja **suma()** przed funkcją **main()**, (b) funkcja **suma()** za funkcją **main()**

Definicję funkcji `suma()` można umieścić także po definicji funkcji `main()`. Ponieważ zasięg widzialności funkcji rozpoczyna się od miejsca jej deklaracji, należy przed definicją funkcji `main()` podać formalną deklarację czyli **prototyp** funkcji `suma()` (Rys. 3b). Prototyp opisuje to samo co nagłówek, ale kończy się średnikiem. Dzięki prototypom kompilator ma możliwość sprawdzenia zgodności typów argumentów wywołania i parametrów funkcji.

```
Program zawierający prototyp funkcji suma().

#include <stdio.h>

float suma(float a, float b);

int main(void)
{
    float x1 = 10.0, x2 = 20.0, wynik;

    wynik = suma(x1,x2);
    printf("Wynik = %f\n", wynik);

    return 0;
}

float suma(float a, float b)
{
    float y;
    y = a + b;
    return y;
}
```

W prototypie nie musimy podawać nazw parametrów - wystarczą tylko typy:

```
float suma(float, float);
```

Jednakże podanie nazw parametrów wpływa na czytelność kodu programu.

2.4. Klasyfikacja funkcji

Funkcja w języku C może zwracać wartość lub jej nie zwracać. Do funkcji mogą być przekazywane argumenty lub też może ich nie być (funkcja bezargumentowa). Z powyższych względów wyróżnia się cztery typy funkcji.

Funkcje nie zwracające wartości i nie mające argumentów

- w nagłówku funkcji, jako typ zwracanej wartości, podaje się słowo **void**,
- w nagłówku funkcji, w miejscu listy jej parametrów, podaje się słowo **void** (tak zaleca standard języka C) lub nie wpisuje się nic,
- jeśli w ciele funkcji występuje **return**, to nie może znajdować się za nim żadna wartość,
- jeśli w ciele funkcji nie występuje **return**, to sterowanie wraca do punktu wywołania na skutek zakończenia wykonywania wszystkich instrukcji znajdujących się w funkcji,
- definicja funkcji może mieć jedną z poniższych postaci:

```
void nazwa(void)
{
    instrukcje;
    return;
}
```

```
void nazwa()
{
    instrukcje;
    return;
}
```

```
void nazwa(void)
{
    instrukcje;
}
```

```
void nazwa()
{
    instrukcje;
}
```

- w wywołaniu takiej funkcji podaje się jej nazwę i nawiasy ():

```
nazwa();
```

Przykład programu zawierającego funkcję, która nie zwraca wartości i nie ma argumentów wywołania:

```
#include <stdio.h>

void drukuj_linie(void)
{
    printf("-----\n");
}

int main(void)
{
    drukuj_linie();
    printf(" Funkcje nie sa trudne!\n");
    drukuj_linie();

    return 0;
}
```

Wynik uruchomienia programu:

```
-----
Funkcje nie sa trudne!
-----
```

Funkcje nie zwracające wartości i mające argumenty

- w nagłówku funkcji, jako typ zwracanej wartości, podaje się słowo **void**,
- jeśli w ciele funkcji występuje **return**, to nie może znajdować się za nim żadna wartość,
- jeśli w ciele funkcji nie występuje **return**, to sterowanie wraca do punktu wywołania na skutek zakończenia wykonywania wszystkich instrukcji znajdujących się w funkcji,
- definicja funkcji może mieć jedną z poniższych postaci:

```
void nazwa (parametry)
{
    instrukcje;
    return;
}
```

```
void nazwa (parametry)
{
    instrukcje;
}
```

- wywołanie funkcji:

```
nazwa (argumenty);
```

Przykład programu zawierającego funkcję, która nie zwraca wartości i ma argumenty wywołania:

```
#include <stdio.h>

void drukuj_dane(char *imie, char *nazwisko, int wiek)
{
    printf("Imie:           %s\n", imie);
    printf("Nazwisko:        %s\n", nazwisko);
    printf("Wiek:              %d\n", wiek);
    printf("Rok urodzenia:    %d\n\n", 2018-wiek);
}

int main(void)
{
    drukuj_dane("Jan", "Kowalski", 23);
    drukuj_dane("Barbara", "Nowak", 28);

    return 0;
}
```

Wynik uruchomienia programu:

```
Imie:           Jan
Nazwisko:        Kowalski
Wiek:           23
Rok urodzenia:  1995

Imie:           Barbara
Nazwisko:        Nowak
Wiek:           28
Rok urodzenia:  1990
```

Funkcje zwracające wartość i nie mające argumentów

- w nagłówku funkcji, w miejscu listy jej parametrów, podaje się słowo **void** (tak zaleca standard języka C) lub nie wpisuje się nic,

- typ wartości zwracanej przez funkcję musi być zgodny z typem wartości występującej po słowie **return**,
- definicja funkcji może mieć jedną z poniższych postaci:

```

typ nazwa(void)
{
    instrukcje;
    return wartość;
}

```

```

typ nazwa()
{
    instrukcje;
    return wartość;
}

```

- w wywołaniu funkcji zwracana wartość podstawiana jest pod zmienną:

```

typ zmienna;
zmienna = nazwa();

```

- poprawne jest także wywołanie funkcji bez podstawiania zwracanej wartości pod zmienną:

```

nazwa();

```

Przykład programu zawierającego funkcję, która zwraca wartość i nie ma argumentów wywołania:

```

#include <stdio.h>

int liczba_sekund_rok(void)
{
    return (365 * 24 * 60 * 60);
}

int main(void)
{
    int wynik = liczba_sekund_rok();
    printf("W roku jest: %d sekund\n", wynik);

    return 0;
}

```

Wynik uruchomienia programu:

W roku jest: 31536000 sekund

Funkcje zwracające wartość i mające argumenty

- najbardziej popularny typ funkcji,
- typ wartości zwracanej przez funkcję musi być zgodny z typem wartości występującej po słowie **return**,
- definicja funkcji powinna mieć postać:

```

typ nazwa(parametry)
{
    instrukcje;
    return wartość;
}

```

- w wywołaniu funkcji zwracana wartość podstawiana jest pod zmienną:

```

typ zmienna;
zmienna = nazwa(argumenty);

```

- poprawne jest także wywołanie funkcji bez podstawiania zwracanej wartości pod zmienną:

```

nazwa(argumenty);

```

Przykład programu zawierającego funkcję, która zwraca wartość i ma argumenty wywołania:

```

#include <stdio.h>

float prad(float nap, float rez)
{
    return (nap/rez);
}

```

```

int main(void)
{
    float U, R, I;

    printf("Podaj U [V]: ");
    scanf("%f", &U);

    printf("Podaj R [Om]: ");
    scanf("%f", &R);

    I = prad(U,R);
    printf("Prad [A]:      %g\n", I);

    return 0;
}

```

Wynik uruchomienia programu:

```

Podaj U [V]: 230
Podaj R [Om]: 150
Prad [A]:      1.53333

```

3. Przebieg ćwiczenia

Na pracowni specjalistycznej należy wykonać wybrane zadania wskazane przez prowadzącego zajęcia. W różnych grupach mogą być wykonywane różne zadania.

1. Napisz program zawierający funkcję wyświetlającą na ekranie wizytówkę o poniższej postaci. Wpisz w wizytówce swoje dane. Wywołaj napisaną funkcję.

```

*****
*           Jan Kowalski           *
* e-mail: j.kowalski@gmail.com    *
*           tel. 123-456-789      *
*****

```

2. Energię elektryczną W pobraną w czasie t przez odbiornik o mocy P określa wzór:

$$W = P \cdot t \quad (1)$$

Napisz funkcję obliczającą i zwracającą zużycie energii elektrycznej (w kWh) pobranej przez odbiornik o mocy P w czasie t . W funkcji `main()` wczytaj z klawiatury wartości P i t , wywołaj napisaną funkcję, a następnie wyświetl wartość przez nią zwróconą.

3. Napisz funkcję zamieniającą odległość podaną w kilometrach na mile lądowe i funkcję zamieniającą odległość podaną w kilometrach na mile morskie. W funkcji `main()` wczytaj z klawiatury odległość w kilometrach, wywołaj napisane funkcje i wyświetl wartości przez nie zwrócone. Uwaga: 1 mila lądowa = 1609,344 metrów, 1 mila morska = 1851,852 metrów.
4. Napisz funkcję obliczającą i zwracającą rezystancję R jednorodnego przewodnika o przekroju poprzecznym S i długości l wykonanego z materiału o rezystywności ρ . Stosując funkcję oblicz rezystancję R przewodnika o długości $l = 100$ m i przekroju $S = 2,5$ mm² w przypadku, gdy jest on wykonany z miedzi, aluminium, srebra lub złota.

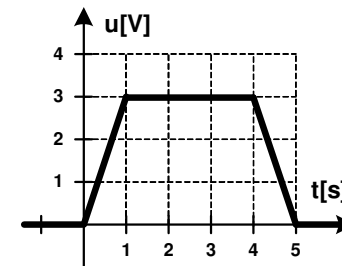
Tabela 1. Rezystywność wybranych materiałów w temperaturze 20 °C

Material	Rezystywność [$\Omega \cdot m$]
miedź	$1,72 \cdot 10^{-8}$
aluminium	$2,82 \cdot 10^{-8}$
srebro	$1,59 \cdot 10^{-8}$
złoto	$2,44 \cdot 10^{-8}$

5. Napisz program zawierający funkcję obliczającą i zwracającą częstotliwość rezonansową f_r układu o rezystancji R , indukcyjności L i pojemności C wprowadzonych z klawiatury w funkcji `main()`.

Przykładowe uruchomienie programu	Wzór
Rezystancja R [Om]: 100 Indukcyjność L [H]: 0.04 Pojemność C [F]: 2.0e-6 ----- Częstotliwość fr [Hz]: 562.697693	$f_r = \frac{1}{2\pi\sqrt{LC}} \quad (2)$

Rezystancja R [Om]: 5000 Indukcyjność L [H]: 0.02 Pojemność C [F]: 4.0e-5 ----- Częstotliwość f_r [Hz]: 177.942413	$f_r = \frac{1}{2\pi\sqrt{LC - \left(\frac{L}{R}\right)^2}} \quad (3)$
Rezystancja R [Om]: 500 Indukcyjność L [H]: 0.03 Pojemność C [F]: 6.0e-5 ----- Częstotliwość f_r [Hz]: 118.508408	$f_r = \frac{1}{2\pi\sqrt{LC - (RC)^2}} \quad (4)$
Rezystancja R [Om]: 10 Indukcyjność L [H]: 1 Pojemność C [F]: 1.0e-6 ----- Częstotliwość f_r [Hz]: 159.146988	$f_r = \frac{1}{2\pi\sqrt{LC - \left(\frac{R}{L}\right)^2}} \quad (5)$
Rezystancja R [Om]: 100 Indukcyjność L [H]: 0.05 Pojemność C [F]: 5.0e-3 ----- Częstotliwość f_r [Hz]: 10.060807	$f_r = \frac{1}{2\pi\sqrt{LC}} \sqrt{1 - \frac{L}{R^2C}} \quad (6)$
Rezystancja R [Om]: 10 Indukcyjność L [H]: 0.1 Pojemność C [F]: 1.0e-6 ----- Częstotliwość f_r [Hz]: 503.54397	$f_r = \frac{1}{2\pi\sqrt{LC - (RC)^2}} \quad (7)$



Rys. 4. Przebieg impulsu trapezowego

6. Suma poniższego szeregu liczbowego wynosi $1/4$. Napisz program zawierający funkcję obliczającą i zwracającą sumę n -wyrzów tego szeregu. Następnie wykorzystując powyższą funkcję wyświetl różnice pomiędzy sumą dokładną ($1/4$), a sumą $n = 10$, $n = 100$ i $n = 1000$ wyrzów tego szeregu.

$$\frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \dots + \frac{1}{n(n+1)(n+2)} \quad (8)$$

7. Rys. 4 przedstawia przebieg impulsu trapezowego. Napisz funkcję, która na podstawie przekazanego do niej czasu t oblicza i zwraca odpowiadającą mu wartość napięcia u . Następnie wykorzystując powyższą funkcję oblicz i wyświetl wartości napięcia dla czasu t równego: **0.0, 0.5, 1.0, ..., 5.5, 6.0 [s]** (zastosuj pętlę **for**).

4. Literatura

- [1] Prata S.: Język C. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2016.
- [2] Kernighan B.W., Ritchie D.M.: Język ANSI C. Programowanie. Wydanie II. Helion, Gliwice, 2010.
- [3] Prinz P., Crawford T.: Język C w pigułce. APN Promise, Warszawa, 2016.
- [4] King K.N.: Język C. Nowoczesne programowanie. Wydanie II. Helion, Gliwice, 2011.
- [5] Kochan S.G.: Język C. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2015.

5. Pytania kontrolne

1. Opisz ogólną strukturę definicji funkcji w języku C.
2. Omów sposób wykonania programu składającego się z więcej niż jednej definicji funkcji.
3. Wyjaśnij czym różni się deklaracja od definicji funkcji?
4. Omów klasyfikację funkcji ze względu na liczbę parametrów i zwracaną wartość.

6. Wymagania BHP

Warunkiem przystąpienia do praktycznej realizacji ćwiczenia jest zapoznanie się z instrukcją BHP i instrukcją przeciw pożarową oraz przestrzeganie zasad w nich zawartych.

W trakcie zajęć laboratoryjnych należy przestrzegać następujących zasad.

- Sprawdzić, czy urządzenia dostępne na stanowisku laboratoryjnym są w stanie kompletnym, nie wskazującym na fizyczne uszkodzenie.
- Jeżeli istnieje taka możliwość, należy dostosować warunki stanowiska do własnych potrzeb, ze względu na ergonomię. Monitor komputera ustawić w sposób zapewniający stałą i wygodną obserwację dla wszystkich członków zespołu.
- Sprawdzić prawidłowość połączeń urządzeń.
- Załączenie komputera może nastąpić po wyrażeniu zgody przez prowadzącego.
- W trakcie pracy z komputerem zabronione jest spożywanie posiłków i picie napojów.
- W przypadku zakończenia pracy należy zakończyć sesję przez wydanie polecenia wylogowania. Zamknięcie systemu operacyjnego może się odbywać tylko na wyraźne polecenie prowadzącego.
- Zabronione jest dokonywanie jakichkolwiek przełączeń oraz wymiana elementów składowych stanowiska.
- Zabroniona jest zmiana konfiguracji komputera, w tym systemu operacyjnego i programów użytkowych, która nie wynika z programu zajęć i nie jest wykonywana w porozumieniu z prowadzącym zajęcia.
- W przypadku zaniku napięcia zasilającego należy niezwłocznie wyłączyć wszystkie urządzenia.
- Stwierdzone wszelkie braki w wyposażeniu stanowiska oraz nieprawidłowości w funkcjonowaniu sprzętu należy przekazywać prowadzącemu zajęcia.

- Zabrania się samodzielnego włączania, manipulowania i korzystania z urządzeń nie należących do danego ćwiczenia.
- W przypadku wystąpienia porażenia prądem elektrycznym należy niezwłocznie wyłączyć zasilanie stanowiska. Przed odłączeniem napięcia nie dotykać porażonego.