Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 2

Kod przedmiotu: **EZ1E3012**

(studia niestacjonarne)

JĘZYK C - FUNKCJE, PRZEKAZYWANIE ARGUMENTÓW DO FUNKCJI, REKURENCJA

Numer ćwiczenia

INF25Z

Autor:
dr inż. Jarosław Forenc

Białystok 2017

Spis treści

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

1. Opis stanowiska

1.1. Stosowana aparatura

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

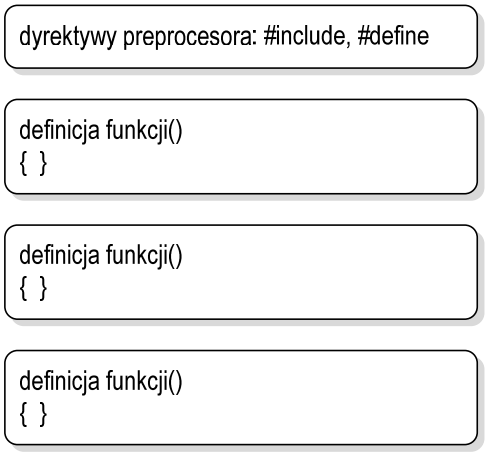
1.2. Oprogramowanie

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).

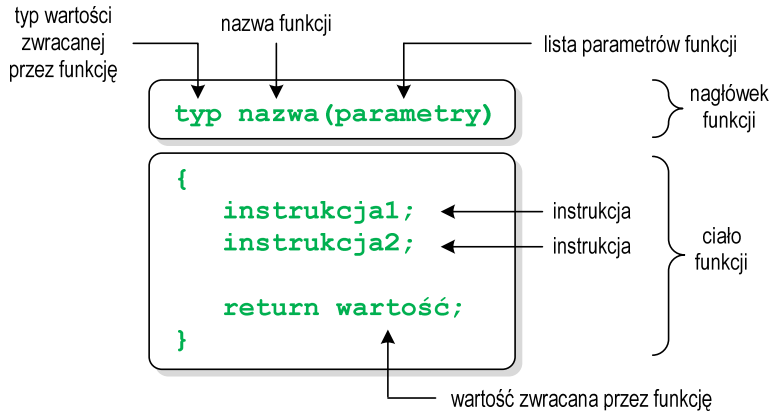


Rys. 1. Struktura programu w języku C

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 **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 **{** i **}**. Pomiędzy 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:

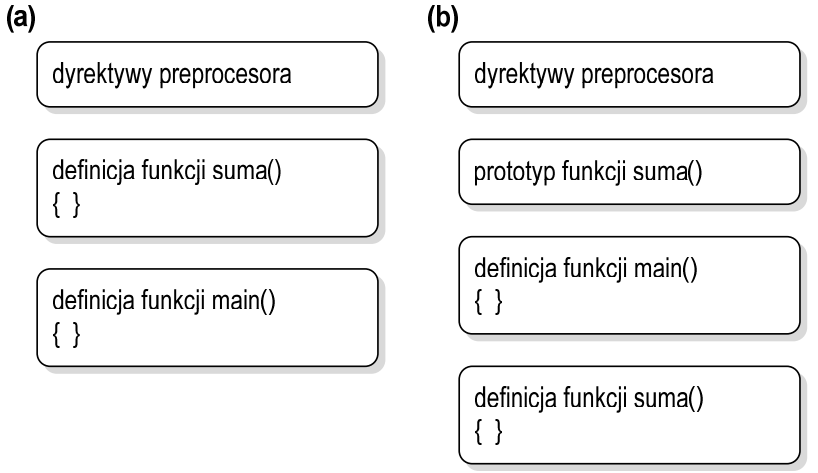

```
float suma(float a, float b)
{
    return (a+b);
}
```

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).



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:

```c
#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:

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

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


- wywołanie funkcji:

```c
nazwa(argumenty);
```

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

```c
#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);
}
```

```c
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
```

2.5. Argumenty i parametry funkcji

Według standardu języka C **argumentem** funkcji nazywa się wyrażenie występujące na, oddzielonej przecinkami i ograniczonej zwykłymi nawiasami, liście wartości przekazywanych do funkcji, umieszczonej w jej wywołaniu. Natomiast **parametrem** funkcji nazywa się obiekt występujący w jej definicji (w nagłówku funkcji), który otrzymuje wartość odpowiedniego argumentu wywołania funkcji.

```c
#include <stdio.h>

void drukuj_punkt(int x, int y)
{
    printf("(%d,%d)",x,y);
}

int main(void)
{
    int x = 10, y = 10;

    drukuj_punkt(x,y);

    return 0;
}
```

parametr funkcji

parametr funkcji

argument funkcji

argument funkcji

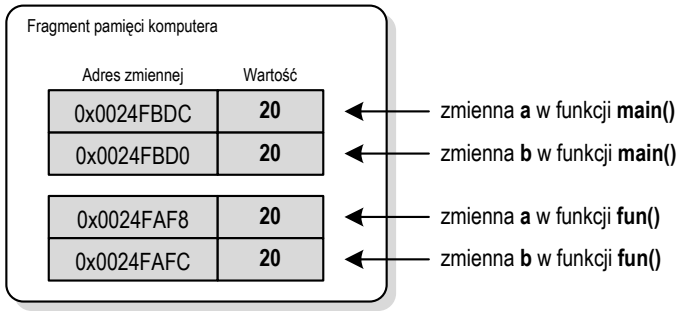
2.6. Przekazywanie argumentów do funkcji przez wartość

Przekazywanie argumentów do funkcji przez wartość oznacza, że po wywołaniu funkcji tworzone są lokalne kopie zmiennych skojarzonych z jej argumentami. W funkcji widoczne są one pod postacią parametrów funkcji. Parametry te mogą być traktowane jak lokalne zmienne, którym przypisano początkową wartość.

Przekazywanie argumentów do funkcji przez wartość.

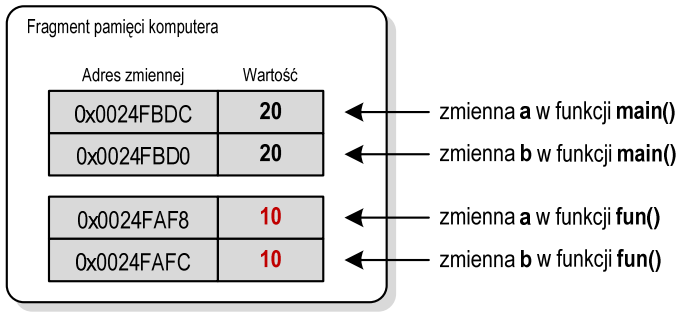
```c
#include <stdio.h>

void fun(int a, int b)
{
    printf("fun1:  a = %d, b = %d\n",a,b);
    a = 10;
    b = 10;
    printf("fun2:  a = %d, b = %d\n",a,b);
}

int main(void)
{
    int a = 20;
    int b = 20;

    printf("main1: a = %d, b = %d\n",a,b);
    fun(a,b);
    printf("main2: a = %d, b = %d\n",a,b);

    return 0;
}
```

W powyższym programie do funkcji **fun()** przekazywane są dwa argumenty (**a** i **b**) zainicjalizowane wartością **20**. Po przekazaniu sterowania do funkcji tworzone są w niej kopie argumentów wywołania. Mają one takie same nazwy (**a** i **b**) i otrzymują także wartość **20**. Fragment pamięci komputera po wejściu do funkcji **fun()** przedstawia Rys. 4.



Rys. 4. Fragment pamięci komputera po wejściu do funkcji **fun()**

Następnie zmiennym **a** i **b** w funkcji **fun()** nadawana jest wartość **10**. Fragment pamięci komputera przed zakończeniem funkcji **fun()** przedstawia Rys. 5.



Rys. 5. Fragment pamięci komputera przed wyjściem z funkcji **fun()**

Po powrocie z funkcji **fun()** zmienne **a** i **b** w funkcji **main()** mają niezmienioną wartość **20**. Stało się tak, gdyż w funkcji **fun()** operacje były wykonywane na ich kopiach. Wynikiem wykonania powyższego programu jest wyświetlenie wartości zmiennych **a** i **b** w różnych fazach jego pracy:

```
main1: a = 20, b = 20
fun1:  a = 20, b = 20
fun2:  a = 10, b = 10
main2: a = 20, b = 20
```

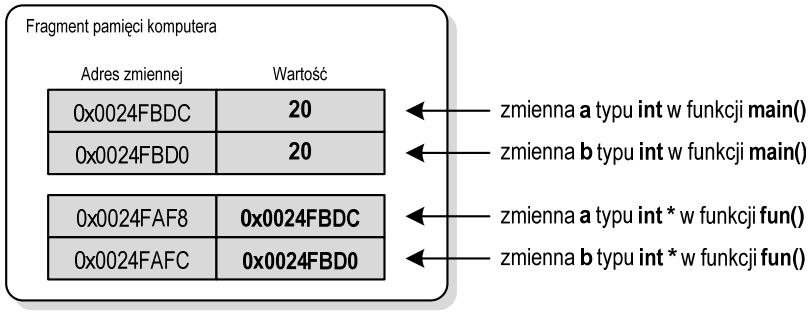

2.7. Przekazywanie argumentów do funkcji przez wskaźnik

Przekazywanie argumentów do funkcji przez wskaźnik polega na tym, że do funkcji przekazywane są adresy zmiennych będących jej argumentami. Wszystkie operacje wykonywane w funkcji na takich argumentach będą odnosiły się do zmiennych z funkcji wywołującej.

Przekazywanie argumentów do funkcji przez wskaźnik.

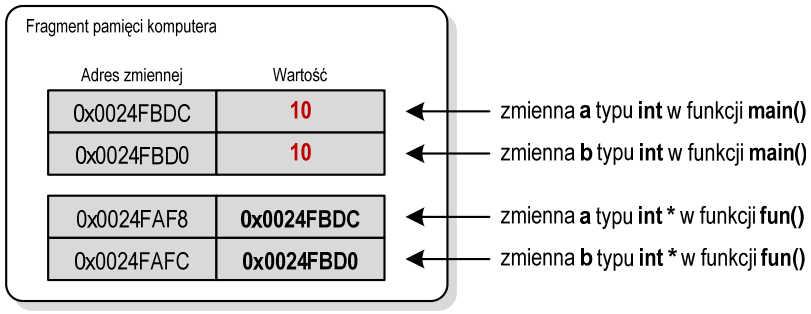
```c
#include <stdio.h>

void fun(int *a, int *b)
{
    printf("fun1:  a = %d, b = %d\n",*a,*b);
    *a = 10;
    *b = 10;
    printf("fun2:  a = %d, b = %d\n",*a,*b);
}

int main(void)
{
    int a = 20
    int b = 20;

    printf("main1: a = %d, b = %d\n",a,b);
    fun(&a,&b);
    printf("main2: a = %d, b = %d\n",a,b);

    return 0;
}
```

W funkcji **main()** w powyższym programie znajdują się deklaracje dwóch zmiennych **a** i **b** typu **int**. Obie zmienne zostały zainicjalizowane wartością **20**. W wywołaniu funkcji **fun()** przed nazwą każdego argumentu znajduje się operator **&**. Oznacza on, że do funkcji nie są przekazywane wartości zmiennych **a** i **b**, ale ich adresy. W nagłówku funkcji **fun()**, przed parametrem **a** i przed parametrem **b**, został dodany symbol gwiazdki: *. Oznacza on, że **a** i **b** są specjalnymi zmiennymi (**wskaźnikami**), które przechowują adresy zmiennych typu **int**. Fragment pamięci komputera po wejściu do funkcji **fun()** przedstawia Rys. 6.



Rys. 6. Fragment pamięci komputera po wejściu do funkcji **fun()**

Mając adres zmiennej można zmienić jej zawartość. Aby poprzez adres dostać się do zawartości zmiennej, należy przed nazwą dodać symbol gwiazdki: ***a**, ***b**. Symbol ten jest tzw. operatorem wyłuskania (odwołania pośredniego). Stosując tego typu odwołania nadajemy w funkcji **fun()** nowe wartości zmiennym **a** i **b** z funkcji **main()**. Fragment pamięci komputera przed zakończeniem funkcji **fun()** przedstawia Rys. 7.



Rys. 7. Fragment pamięci komputera przed wyjściem z funkcji **fun()**

Wynikiem wykonania powyższego programu jest wyświetlenie wartości zmiennych **a** i **b** w różnych fazach jego pracy:

```
main1: a = 20, b = 20
fun1:  a = 20, b = 20
fun2:  a = 10, b = 10
main2: a = 10, b = 10
```


Zmienne **a** i **b** występujące w funkcjach **fun()** i **main()** mają takie same nazwy, ale inne typy. W funkcji **main()**:

int a; - deklaracja zmiennej typu **int**,
a - zmienna typu **int**,
&a - adres zmiennej **a**, a nie jej wartość.

W funkcji **fun()**:

int *a; - deklaracja zmiennej wskaźnikowej (wskaźnik do typu **int**),
a - adres zmiennej typu **int**,
***a** - wartość zmiennej wskazywanej przez **a**.

Przekazywanie argumentów do funkcji przez wskaźnik stosowane jest wtedy, gdy funkcja powinna zwrócić więcej niż jedną wartość. W poniższym programie znajduje się funkcja rozwiązująca równanie kwadratowe.

Rozwiązanie równania kwadratowego.

```
#include <stdio.h>
#include <math.h>

int rkw(float a, float b, float c, float *x1, float *x2)
{
    float delta;

    delta = b*b - 4*a*c;

    if (delta < 0)
        return 0;
    if (delta == 0)
    {
        *x1 = *x2 = -b/(2*a);
        return 1;
    }
    if (delta > 0)
    {
        *x1 = (-b - sqrt(delta))/(2*a);
        *x2 = (-b + sqrt(delta))/(2*a);
        return 2;
    }
}
```

```c
int main(void)
{
    float a, b, c, x1, x2;

    printf("Podaj a: ");
    scanf("%f",&a);

    printf("Podaj b: ");
    scanf("%f",&b);

    printf("Podaj c: ");
    scanf("%f",&c);

    switch(rkw(a,b,c,&x1,&x2))
    {
    case 0:
        printf("Brak pierwiastkow\n");
        break;
    case 1:
        printf("x1 = x2 = %f\n",x1);
        break;
    case 2:
        printf("x1 = %f\n",x1);
        printf("x2 = %f\n",x2);
    }

    return 0;
}
```

Przykładowy wynik działania powyższego programu:

```
Podaj a: 1
Podaj b: -3
Podaj c: 2
x1 = 1.000000
x2 = 2.000000
```

Do funkcji **rkw()** przekazywanych jest 5 argumentów. Pierwsze trzy (**a**, **b**, **c**) są współczynnikami równania kwadratowego przekazywanymi przez wartość. Natomiast pozostałe dwa argumenty (**x1**, **x2**) są pierwiastkami tego równania przekazywanymi przez wskaźnik. Funkcja **rkw()** oblicza wartości zmiennych **x1** i **x2**, a następnie zwraca liczbę pierwiastków (**0**, **1** lub **2**). Wywołanie funkcji

rkw() zostało umieszczone bezpośrednio w instrukcji **switch**. Zależnie od wartości zwróconej przez funkcję, wyświetlany jest: komunikat o braku pierwiastków, wartość jednego podwójnego pierwiastka (**x1 = x2**) lub wartość dwóch pierwiastków (**x1**, **x2**).

2.8. Przekazywanie tablicy jednowymiarowej do funkcji

Tablica jednowymiarowa (wektor) przekazywana jest do funkcji przez wskaźnik. Nie jest zatem tworzona jej kopia, a wszystkie operacje na jej elementach odnoszą się do tablicy z funkcji wywołującej. W nagłówku funkcji należy podać typ elementów tablicy, jej nazwę oraz nawiasy kwadratowe z liczbą elementów tablicy lub same nawiasy kwadratowe.

```c
void fun(int tab[5])
{
    ...
}
```

lub

```c
void fun(int tab[])
{
    ...
}
```

Jeśli argumentem funkcji jest tablica, to w wywołaniu funkcji podaje się tylko jej nazwę (bez żadnych nawiasów kwadratowych):

```c
fun(tab);
```

W poniższym programie do funkcji **zero()** przekazywana jest tablica jednowymiarowa o nazwie **tab**. Podczas deklaracji tablica ta inicjalizowana jest kolejnymi liczbami: **1, 2, …, 5**. Następnie elementy tablicy wyświetlane są na ekranie. Wywołanie i wykonanie funkcji **zero()** powoduje zapisanie wartości **0** do każdego elementu tablicy. Po powrocie z funkcji **zero()** elementy tablicy wyświetlane są ponownie na ekranie.

Przekazywanie tablicy jednowymiarowej do funkcji.

```c
#include <stdio.h>
```

```c
void zero(int tab[])
{
    int i;

    for (i=0; i<5; i++)
        tab[i] = 0;
}

int main(void)
{
    int tab[5] = {1,2,3,4,5}, i;

    for (i=0; i<5; i++)
        printf("%3d",tab[i]);
    printf("\n");

    zero(tab);

    for (i=0; i<5; i++)
        printf("%3d",tab[i]);
    printf("\n");

    return 0;
}
```

Wynik działania powyższego programu:

```
  1   2   3   4   5
  0   0   0   0   0
```

2.9. Przekazywanie tablicy dwuwymiarowej do funkcji

Tablica dwuwymiarowa (macierz) przekazywana jest do funkcji także przez wskaźnik. W nagłówku funkcji należy podać typ elementów tablicy, jej nazwę oraz w nawiasach kwadratowych liczbę wierszy i kolumn lub tylko liczbę kolumn.

```c
void fun(int tab[2][3])
{
    ...
}
```

lub

```c
void fun(int tab[][3])
{
    ...
}
```


W wywołaniu funkcji podaje się tylko nazwę tablicy (bez nawiasów kwadratowych).

```c
fun(tab);
```

W poniższym programie znajdują się funkcje **zero()** i **drukuj()**, do których przekazywana jest tablica dwuwymiarowa (macierz).

Przekazywanie tablicy dwuwymiarowej do funkcji.

```c
#include <stdio.h>

void zero(int tab[][3])
{
    int i, j;

    for (i=0; i<2; i++)
        for (j=0; j<3; j++)
            tab[i][j] = 0;
}

void drukuj(int tab[2][3])
{
    int i, j;

    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
            printf("%3d",tab[i][j]);
        printf("\n");
    }
    printf("\n");
}

int main(void)
{
    int tab[2][3] = {{1,2,3},{4,5,6}};

    drukuj(tab);
    zero(tab);
    drukuj(tab);

    return 0;
}
```

Wynik działania powyższego programu:

```
1  2  3
4  5  6

0  0  0
0  0  0
```

2.10. Przekazywanie struktur do funkcji

Struktury przekazywane są do funkcji tak samo jak każde inne zmienne podstawowych typów, czyli przez wartość (nawet jeśli daną składową jest tablica).

Program zawierający funkcję obliczającą odległość dwóch punktów.

```
#include <stdio.h>
#include <math.h>

struct punkt
{
    float x;
    float y;
};

float odleglosc(struct punkt pkt1, struct punkt pkt2)
{
    return sqrt(pow(pkt2.x-pkt1.x,2)+
                pow(pkt2.y-pkt1.y,2));
}

int main(void)
{
    struct punkt p1 = {2,3};
    struct punkt p2 = {-2,1};
    float odl;

    odl = odleglosc(p1,p2);

    printf("Punkt nr 1: (%g,%g)\n",p1.x,p1.y);
    printf("Punkt nr 2: (%g,%g)\n",p2.x,p2.y);
    printf("Odleglosc = %g\n",odl);

    return 0;
}
```


Wynik uruchomienia programu będzie następujący:

```
Punkt nr 1: (2,3)
Punkt nr 2: (-2,1)
Odleglosc = 4.47214
```

2.11. Rekurencyjne wywołanie funkcji

Rekurencyjne wywołanie funkcji występuje wtedy, gdy funkcja wywołuje sama siebie. Aby ciąg rekurencyjnych wywołań mógł zakończyć się musi istnieć odpowiedni warunek stopu. Przed wykonaniem kolejnego wywołania funkcja powinna sprawdzić ten warunek. Jeśli będzie on prawdziwy, to nie nastąpi kolejne wywołanie rekurencyjne.

Mechanizm rekurencji jest często stosowany do definiowania i opisywania algorytmów. Przykłady zapisu algorytmów rekurencyjnych w postaci wzorów matematycznych:

- silnia:

$$n! = \begin{cases} 1 & dla \quad n = 0 \\ n(n-1)! & dla \quad n \geq 1 \end{cases} \tag{1}$$

Funkcja obliczająca silnię liczby - wersja 1 (nierekurencyjna).

```
int silnia(int n)
{
    int i, wynik = 1;

    for (i=1; i<=n; i++)
        wynik = wynik * i;

    return wynik;
}
```

Funkcja obliczająca silnię liczby - wersja 2 (rekurencyjna).

```
int silnia(int n)
{
    if (n==0)
        return 1;
    else
        return n*silnia(n-1);
}
```

Funkcja obliczająca silnię liczby - wersja 3 (rekurencyjna).

```
int silnia(int n)
{
    return n ? n*silnia(n-1) : 1;
}
```

- ciąg Fibonacciego:

$$F_n = \begin{cases} 0 & dla \quad n = 0 \\ 1 & dla \quad n = 1 \\ F_{n-1} + F_{n-2} & dla \quad n > 1 \end{cases} \qquad (2)$$

Funkcja obliczająca n-ty wyraz ciągu Fibonacciego (rekurencyjna).

```
int Fibo(int n)
{
    if (n==0)
        return 0;
    else
        if (n==1)
            return 1;
        else
            return Fibo(n-1) + Fibo(n-2);
}
```


- algorytm Euklidesa:

$$NWD(a,b) = \begin{cases} a & dla \quad b = 0 \\ NWD(b, a \bmod b) & dla \quad b \geq 1 \end{cases} \qquad (3)$$

Funkcja obliczająca największy wspólny dzielnik dwóch liczb (rekurencyjna).

```
int NWD(int a, int b)
{
    return b ? NWD(b,a%b) : a;
}
```

Poniższy program przedstawia sposób wywołania rekurencyjnej funkcji **NWD()**.

Obliczenie największego wspólnego dzielnika dwóch liczb.

```
#include <stdio.h>
#include <math.h>

int NWD(int a, int b)
{
    return b ? NWD(b,a%b) : a;
}

int main(void)
{
    int a, b, nwd;

    printf("Podaj a: ");
    scanf("%d",&a);

    printf("Podaj b: ");
    scanf("%d",&b);

    nwd = NWD(a,b);

    printf("NWD(%d,%d) = %d\n",a,b,nwd);

    return 0;
}
```

Przykładowy wynik działania powyższego programu:

```
Podaj a: 4368
Podaj b: 3584
NWD(4368,3584) = 112
```

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:

$$W = P \cdot t \tag{4}$$

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

Przykładowe uruchomienie programu	Wzór

`Rezystancja R [Om]: 100`
`Indukcyjnosc L [H]: 0.04`
`Pojemnosc C [F]: 2.0e-6`
`----------`
`Czestotliwosc fr [Hz]: 562.697693`
	$f_r = \frac{1}{2\pi\sqrt{LC}}$ (5)

`Rezystancja R [Om]: 5000`
`Indukcyjnosc L [H]: 0.02`
`Pojemnosc C [F]: 4.0e-5`
`----------`
`Czestotliwosc fr [Hz]: 177.942413`
	$f_r = \frac{1}{2\pi\sqrt{LC - \left(\frac{L}{R}\right)^2}}$ (6)

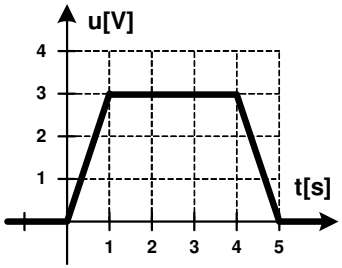
`Rezystancja R [Om]: 500`
`Indukcyjnosc L [H]: 0.03`
`Pojemnosc C [F]: 6.0e-5`
`----------`
`Czestotliwosc fr [Hz]: 118.508408`
	$f_r = \frac{1}{2\pi}\sqrt{\frac{1}{LC} - \frac{1}{(RC)^2}}$ (7)

`Rezystancja R [Om]: 10`
`Indukcyjnosc L [H]: 1`
`Pojemnosc C [F]: 1.0e-6`
`----------`
`Czestotliwosc fr [Hz]: 159.146988`
	$f_r = \frac{1}{2\pi}\sqrt{\frac{1}{LC} - \left(\frac{R}{L}\right)^2}$ (8)

`Rezystancja R [Om]: 100`
`Indukcyjnosc L [H]: 0.05`
`Pojemnosc C [F]: 5.0e-3`
`----------`
`Czestotliwosc fr [Hz]: 10.060807`
	$f_r = \frac{1}{2\pi\sqrt{LC}}\sqrt{1 - \frac{L}{R^2C}}$ (9)

`Rezystancja R [Om]: 10`
`Indukcyjnosc L [H]: 0.1`
`Pojemnosc C [F]: 1.0e-6`
`----------`
`Czestotliwosc fr [Hz]: 503.54397`
	$f_r = \frac{1}{2\pi\sqrt{LC - (RC)^2}}$ (10)

5. Rys. 8 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**).



Rys. 8. Przebieg impulsu trapezowego

6. Napisz program zawierający funkcje wykonujące operacje na **N** - elementowym wektorze liczb całkowitych:

   - **generuj()** - funkcja zapisująca do wektora wygenerowane pseudolosowo liczby całkowite z zakresu ⟨**a, b**⟩, gdzie **a** i **b** są argumentami funkcji;
   - **wyswietl()** - funkcja wyświetlająca elementy wektora w jednym wierszu;
   - **suma()** - funkcja zwracająca sumę elementów wektora;
   - **odwroc()** - funkcja odwracająca kolejność elementów w wektorze.

   Wywołaj w programie wszystkie zdefiniowane funkcje.

7. Napisz funkcję, do której przekazywany jest **N** - elementowy wektor liczb całkowitych oraz liczba całkowita **x**. Funkcja powinna obliczyć i zwrócić ilość wystąpień liczby **x** w wektorze. Elementy wektora wygeneruj pseudolosowo i wyświetl je na ekranie.

8. Napisz funkcję, do której przekazywana jest **N×M** - elementowa tablica liczb całkowitych. Funkcja powinna **odwrócić kolejność elementów** w poszczególnych wierszach tablicy.

9. Napisz program zawierający funkcje realizujące podstawowe operacje arytmetyczne na liczbach zespolonych (+, -, *, /) oraz funkcję wyświetlającą liczbę zespoloną w postaci: 2+3j. Liczbę zespoloną zdefiniuj jako strukturę:


```
struct zesp
{
   float Re, Im;
};
```

Nagłówki pozostałych funkcji powinny mieć postać:

```
struct zesp dodaj (struct zesp x, struct zesp y);
struct zesp odejmij (struct zesp x, struct zesp y);
struct zesp pomnoz (struct zesp x, struct zesp y);
struct zesp podziel (struct zesp x, struct zesp y);
void drukuj (struct zesp x);
```

Wywołaj zdefiniowane funkcje dla podanych liczb zespolonych:

z1 = 2 + j4 z2 = 6 - j8

Sprawdź poprawność otrzymanych wyników z tabelą 1.

Tabela 1. Poprawne wyniki do zadania 9

Operacja	Wynik
z1 + z2	8 - j4
z1 - z2	-4 + j12
z1 * z2	44 + j8
z1 / z2	-0,2 + j0,4

10. Napisz funkcję rekurencyjną obliczającą $\mathbf{x}^{\mathbf{n}}$ na podstawie wzoru:

$$x^n = x \cdot x^{n-1} \tag{11}$$

Wczytaj **x** (liczba rzeczywista) i **n** (liczba naturalna) z klawiatury.

11. Ciąg liczbowy opisany jest podanym wzorem rekurencyjnym:

$$a_n = \begin{cases} 2{,}5 & \text{dla} \quad n = 0 \\ 3{,}5 & \text{dla} \quad n = 1 \\ 1{,}5 \cdot a_{n-1} + 2{,}5 \cdot a_{n-2} & \text{dla} \quad n > 1 \end{cases} \tag{12}$$

Napisz funkcję rekurencyjną obliczającą n-ty wyraz tego ciągu. Stosując napisaną funkcję oblicz i wyświetl sumę wyrazów od a_0 do a_5.

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.

[6] Reese R.: Wskaźniki w języku C. Przewodnik. Helion, Gliwice, 2014.

[7] Reek K.A.: Język C. Wskaźniki. Vademecum profesjonalisty. Helion, Gliwice, 2003.

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.
5. Wyjaśnij różnice w przekazywaniu parametrów do funkcji przez wartość i wskaźnik.
6. Opisz sposób przekazywania do funkcji tablic jedno- i dwuwymiarowych oraz struktur.
7. Co to jest rekurencyjne wywołanie funkcji i kiedy jest stosowane?

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.