



Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

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 2021

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.....	7
2.4. Klasyfikacja funkcji	9
2.5. Argumenty i parametry funkcji	14
2.6. Przekazywanie argumentów do funkcji przez wartość	15
2.7. Przekazywanie argumentów do funkcji przez wskaźnik.....	17
2.8. Przekazywanie tablicy jednowymiarowej do funkcji	22
2.9. Przekazywanie tablicy dwuwymiarowej do funkcji	23
2.10. Przekazywanie struktury do funkcji.....	25
2.11. Rekurencyjne wywołanie funkcji	26
3. Przebieg ćwiczenia.....	29
4. Literatura.....	33
5. Pytania kontrolne	34
6. Wymagania BHP	34

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

© Wydział Elektryczny, Politechnika Białostocka, 2021 (wersja 5.0)

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 10.

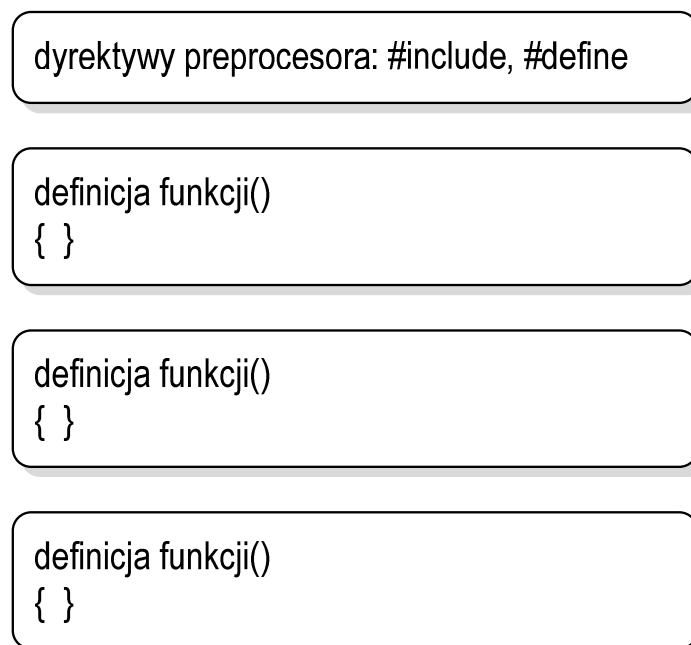
1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Code::Blocks.

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).

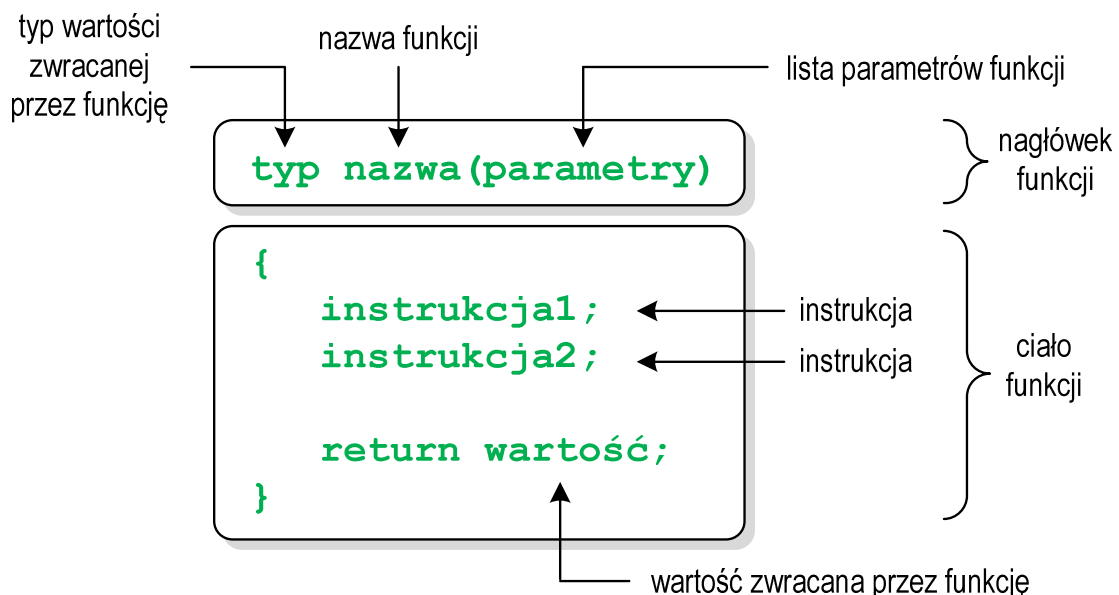


Rys. 1. Struktura programu w języku C

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: **{ i }**. Pomędzy 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 i zwracająca 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, x2 = 20, 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**). Parametry funkcji traktowane są tak samo jak zmienne zadeklarowane w tej funkcji i zainicjalizowane wartościami argumentów wywołania. Jeśli funkcja ma kilka parametrów, to dla każdego z nich podaje się typ i nazwę parametru. Parametry oddzielane są od siebie przecinkami.

W ciele funkcji można umieszczać dowolne konstrukcje języka C: deklaracje zmiennych typów prostych i złożonych, instrukcje warunkowe, pętle, itp. W powyższym programie funkcja **suma()** zawiera instrukcję, w której dodawane są do siebie zmienne **a** i **b**, zaś wynik zapisywany jest w zmiennej **y**.

Powrót z funkcji (do miejsca zaraz po jej wywołaniu) następuje na skutek wykonania instrukcji **return**. Wartość zwracana przez funkcję (czyli wartość zmiennej **y** występującej bezpośrednio po **return**) 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, pomijając zmienną **y**.

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

W definicji funkcji można nadać jej parametrom domyślne wartości.

```
float suma(float a = 15, float b = 25)
{
    return (a+b);
}
```

W takim przypadku funkcję można wywołać z dwoma, jednym lub bez żadnych argumentów. Brakujące argumenty zostaną zastąpione wartościami domyślnymi.

```
wynik = suma(x1, x2);    // 10 + 20
wynik = suma(x1);      // 10 + 25
wynik = suma();        // 15 + 25
```

Nie wszystkie parametry muszą mieć podane domyślne wartości. Wartości muszą być podawane od prawej strony listy parametrów.

```
float suma(float a, float b = 25)
{
    return (a+b);
}
```

Powyższa funkcja może być wywołana z dwoma lub jednym argumentem.

```
wynik = suma(x1, x2); // 10 + 20
wynik = suma(x1); // 10 + 25
```

W wywołaniu funkcji jako argumenty mogą występować stałe liczbowe, nazwy zmiennych, wyrażenia arytmetyczne lub wywołania innych funkcji. Wywołanie funkcji może być argumentem innej 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 funkcji, ale kończy się średnikiem. Dzięki prototypom kompilator sprawdza w wywołaniu funkcji: nazwę funkcji, liczbę i typ argumentów, typ zwracanej wartości.

Program zawierający prototyp funkcji `suma()`.

```
#include <stdio.h>

float suma(float a, float b);

int main(void)
{
    float x1 = 10, x2 = 20, 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);
```

Podanie nazw parametrów wpływa na czytelność kodu programu.

W przypadku umieszczenia prototypu funkcji i pominięcia jej definicji błąd wystąpi nie na etapie kompilacji programu, ale na etapie łączenia (linkowania).

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.

Funkcja nie zwracająca wartości i nie posiadająca 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 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!
-----
```

Funkcja nie zwracająca wartości i posiadająca 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", 2021-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:  1998

Imie:           Barbara
Nazwisko:        Nowak
Wiek:           28
Rok urodzenia:  1993
```

Funkcja zwracająca wartość i nie posiadająca 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

Funkcja zwracająca wartość i posiadająca 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 moc)
{
    return (moc/nap);
}
```

```

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

    printf("Podaj U [V]:  "); scanf("%f", &U);
    printf("Podaj P [W]:  "); scanf("%f", &P);
    I = prad(U,P);
    printf("Prad [A]:      %g\n", I);

    return 0;
}

```

Wynik uruchomienia programu:

```

Podaj U [V]:  230
Podaj P [W]:  1600
Prad [A]:      6.95652

```

2.5. Argumenty i parametry funkcji

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

```

#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;
}

```

Diagram illustrating function arguments and parameters:

- Arrows point from the text "parametr funkcji" to the variables `x` and `y` in the function signature `void drukuj_punkt (int x, int y)`.
- Arrows point from the text "argument funkcji" to the variables `x` and `y` in the function call `drukuj_punkt (x, y);`.

2.6. Przekazywanie argumentów do funkcji przez wartość

W języku C istnieją dwie metody przekazywania argumentów do funkcji: przez wartość i przez wskaźnik.

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.

```
#include <stdio.h>

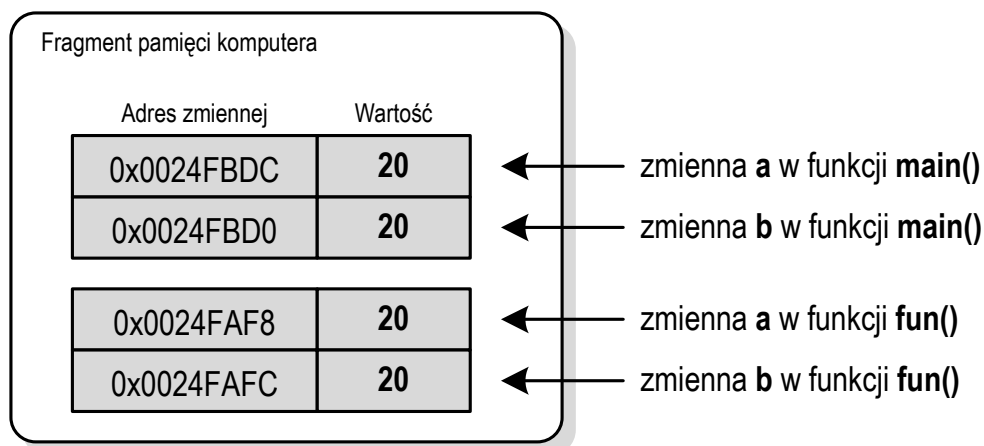
void fun(int a, int b)
{
    printf("fun1:  a = %d, b = %d\n", a,b);
    a = 10;
    b = 10;
    printf("fun2:  a = %d, b = %d\n", a,b);
}

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);

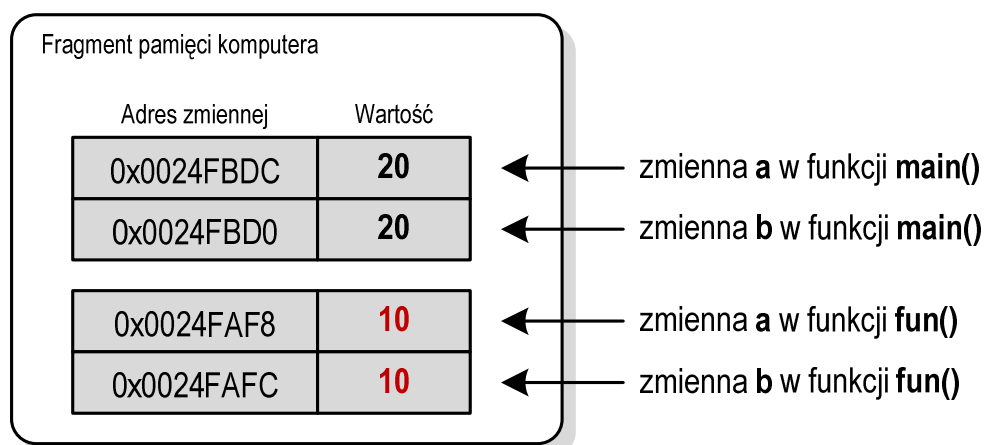
    return 0;
}
```

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**, **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 niezmienną 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.

```
#include <stdio.h>

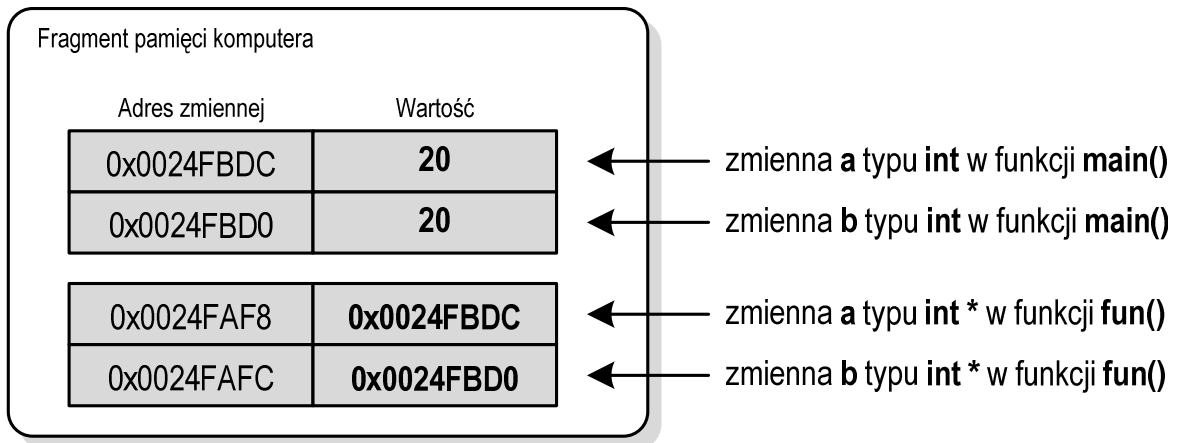
void fun(int *a, int *b)
{
    printf("fun1:  a = %d, b = %d\n", *a, *b);
    *a = 10;
    *b = 10;
    printf("fun2:  a = %d, b = %d\n", *a, *b);
}

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);

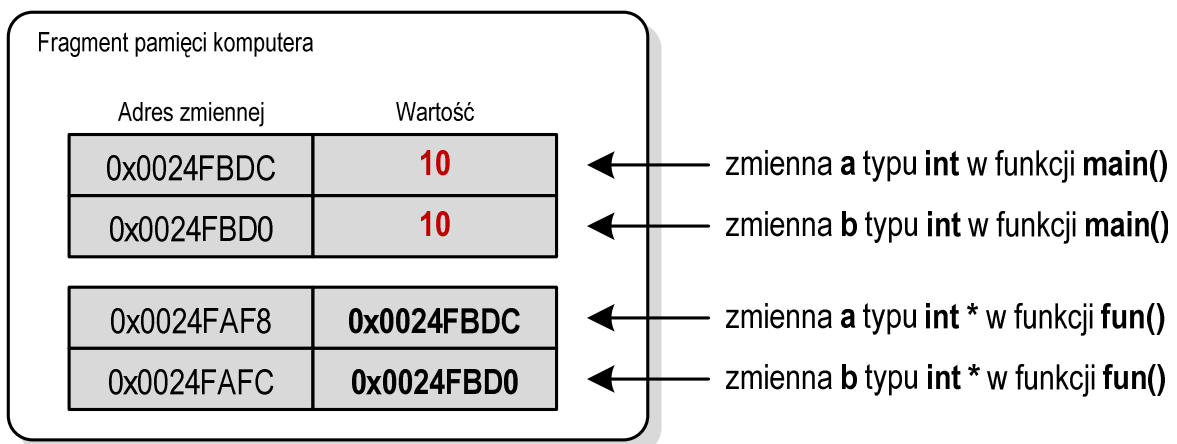
    return 0;
}
```

W funkcji `main()` 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 wartość. Aby poprzez adres dostać się do wartości zmiennej, należy przed nazwą dodać symbol gwiazdki: ***a**, ***b**. Symbol ten jest operatorem wyluskania (odwołania pośredniego). Stosując tego typu odwołania nadajemy w funkcji **fun()** nowe wartości zmiennym **a** i **b** z funkcji **main()**. Fragment pamięci 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 **a** typu **int**,
a - zmienna typu **int**,
&a - adres zmiennej **a**.

W funkcji **fun()**:

int *a; - deklaracja zmiennej wskaźnikowej **a** (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 rkq(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;
    }
}
```

```

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 = %g\n", x1);
        break;
    case 2:
        printf("x1 = %g\n", x1);
        printf("x2 = %g\n", x2);
    }

    return 0;
}

```

Przykładowe wyniki uruchomienia programu:

Podaj a: 1	Podaj a: 1	Podaj a: 2
Podaj b: -3	Podaj b: 2	Podaj b: 2
Podaj c: 2	Podaj c: 1	Podaj c: 1
x1 = 1	x1 = x2 = -1	Brak pierwiastkow
x2 = 2		

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`).

W kolejnym przykładzie znajduje się funkcja `swap()`, której zadaniem jest zamiana miejscami wartości dwóch zmiennych.

Zamiana miejscami wartości dwóch zmiennych.

```
#include <stdio.h>

void swap(int *p1, int *p2)
{
    int tmp;

    tmp = *p1;
    *p1 = *p2;
    *p2 = tmp;
}

int main(void)
{
    int a = 5;
    int b = 7;

    printf("Przed:  a = %d, b = %d\n", a, b);

    swap(&a, &b);

    printf("Po:      a = %d, b = %d\n", a, b);

    return 0;
}
```

Wynik uruchomienia programu:

```
Przed:  a = 5, b = 7
Po:      a = 7, b = 5
```

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.

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

lub

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

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

```
fun(tab);
```

Poniższy program zawiera trzy funkcje, do których przekazywana jest tablica jednowymiarowa (wektor). Funkcja **drukuj()** wyświetla elementy tablicy na ekranie, funkcja **zeruj()** zapisuje wartość **0** do każdego elementu tablicy, zaś funkcja **srednia()** oblicza i zwraca średnią arytmetyczną elementów tablicy. W funkcji **main()** tablica inicjalizowana jest liczbami całkowitymi. Następnie obliczana jest średnia arytmetyczna elementów tablicy. W kolejnych instrukcjach elementy tablicy są wyświetlane na ekranie, zerowane i ponownie wyświetlane.

Przekazywanie tablicy jednowymiarowej do funkcji.

```
#include <stdio.h>

void drukuj(int tab[])
{
    for (int i=0; i<5; i++)
        printf("%3d", tab[i]);
    printf("\n");
}
```

```

void zeruj(int tab[5])
{
    for (int i=0; i<5; i++)
        tab[i] = 0;
}

float srednia(int tab[])
{
    int suma = 0;

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

    return (float)suma/5;
}

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

    sr = srednia(tab);
    printf("Srednia = %g\n",sr);

    drukuj(tab);
    zeruj(tab);
    drukuj(tab);

    return 0;
}

```

Wynik uruchomienia programu:

```

Srednia = 3.4
  3  5  6  2  1
  0  0  0  0  0

```

2.9. Przekazywanie tablicy dwuwymiarowej do funkcji

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

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

lub

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

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

```
fun(tab);
```

W poniższym programie znajdują się trzy funkcje, do których przekazywana jest tablica dwuwymiarowa (macierz). Funkcja **drukuj()** wyświetla elementy tablicy na ekranie, funkcja **zeruj()** zapisuje wartość **0** do każdego elementu tablicy, zaś funkcja **srednia()** oblicza i zwraca średnią arytmetyczną elementów tablicy.

Przekazywanie tablicy dwuwymiarowej do funkcji.

```
#include <stdio.h>

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

void zeruj(int tab[][3])
{
    for (int i=0; i<2; i++)
        for (int j=0; j<3; j++)
            tab[i][j] = 0;
}
```



```

float srednia(int tab[][3])
{
    int suma = 0;

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

    return (float)suma/(2*3);
}

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

    sr = srednia(tab);
    printf("Srednia = %g\n\n", sr);

    drukuj(tab);
    zeruj(tab);
    drukuj(tab);

    return 0;
}

```

Wynik uruchomienia programu:

```
Srednia = 3.5
```

```
1 2 3
4 5 6
```

```
0 0 0
0 0 0
```

2.10. Przekazywanie struktury do funkcji

Struktura przekazywana jest do funkcji tak samo jak każda inna zmienna standardowego typu, czyli przez wartość (nawet jeśli daną składową struktury jest tablica). W poniższym programie znajduje się funkcja **odleglosc()**, do której przekazywane są dwie zmienne strukturalne. Funkcja ta oblicza i zwraca odległość dwóch punktów w prostokątnym układzie współrzędnych.

Program zawierający funkcję obliczającą i zwracającą 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:

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

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>=1 ? n*silnia(n-1) : 1;
}
```

- ciąg Fibonacciego:

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

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

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

- algorytm Euklidesa:

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

Funkcja obliczająca największy wspólny dzielnik dwóch liczb (rekurencyjna).

```
int NWD(int a, int b)
{
    return b>=1 ? 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>=1 ? 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 uruchomienia 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 (pamiętaj o ramce z gwiazdek). 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:

$$W = P \cdot t \quad (4)$$

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.

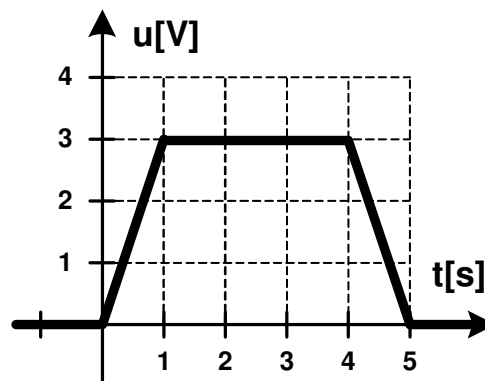
Uwaga: 1 mila lądowa = 1609,344 metrów, 1 mila morska = 1851,852 metrów.

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

Przykładowe uruchomienie programu	Wzór
<pre>Indukcyjnosc L [H]: 0.04 Pojemnosc C [F]: 2.0e-6 ----- Czestotliwosc fr [Hz]: 562.697693</pre>	$f_r = \frac{1}{2\pi\sqrt{LC}} \quad (5)$
<pre>Rezystancja R [Om]: 5000 Indukcyjnosc L [H]: 0.02 Pojemnosc C [F]: 4.0e-5 ----- Czestotliwosc fr [Hz]: 177.942413</pre>	$f_r = \frac{1}{2\pi\sqrt{LC - \left(\frac{L}{R}\right)^2}} \quad (6)$
<pre>Rezystancja R [Om]: 500 Indukcyjnosc L [H]: 0.03 Pojemnosc C [F]: 6.0e-5 ----- Czestotliwosc fr [Hz]: 118.508408</pre>	$f_r = \frac{1}{2\pi\sqrt{\frac{1}{LC} - \frac{1}{(RC)^2}}} \quad (7)$
<pre>Rezystancja R [Om]: 10 Indukcyjnosc L [H]: 1 Pojemnosc C [F]: 1.0e-6 ----- Czestotliwosc fr [Hz]: 159.146988</pre>	$f_r = \frac{1}{2\pi\sqrt{\frac{1}{LC} - \left(\frac{R}{L}\right)^2}} \quad (8)$

Rezystancja R [Om]: 100 Indukcyjność L [H]: 0.05 Pojemność C [F]: 5.0e-3 ----- Częstotliwość fr [Hz]: 10.060807	$f_r = \frac{1}{2\pi\sqrt{LC}} \sqrt{1 - \frac{L}{R^2C}} \quad (9)$
Rezystancja R [Om]: 10 Indukcyjność L [H]: 0.1 Pojemność C [F]: 1.0e-6 ----- Częstotliwość fr [Hz]: 503.54397	$f_r = \frac{1}{2\pi\sqrt{LC - (RC)^2}} \quad (10)$

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 zmieniającego się od 0 do 6 sekund z krokiem 0,25 sekundy (zastosuj pętlę **for**). Wyświetl wyniki w dwóch kolumnach (**czas, napięcie**).



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 $\langle a, b \rangle$, gdzie a i b są argumentami funkcji;
 - wyświetl()** - funkcja wyświetlająca elementy wektora w jednym wierszu;
 - suma()** - funkcja obliczająca i zwracająca sumę elementów wektora;
 - ile_x()** - funkcja zwracającą liczbę wystąpień wartości x w wektorze (x - argument funkcji);

e) **odwroc()** - funkcja odwracająca kolejność elementów w wektorze.

Wywołaj w programie wszystkie zdefiniowane funkcje.

7. Napisz program zawierający funkcje wykonujące operacje na $N \times M$ - elementowej macierzy liczb całkowitych:

a) **generuj()** - funkcja zapisująca do macierzy wygenerowane pseudolosowo liczby całkowite z zakresu $\langle a, b \rangle$, gdzie **a** i **b** są argumentami funkcji;

b) **wyświetl()** - funkcja wyświetlająca elementy macierzy z podziałem na wiersze i kolumny;

c) **suma()** - funkcja obliczająca i zwracająca sumę elementów macierzy;

d) **ile_x()** - funkcja zwracająca liczbę wystąpień wartości **x** w macierzy (**x** - argument funkcji);

e) **odwroc_wiersz()** - funkcja odwracająca kolejność elementów w poszczególnych wierszach macierzy;

f) **odwroc_kolumna()** - funkcja odwracająca kolejność elementów w poszczególnych kolumnach macierzy;

Wywołaj w programie wszystkie zdefiniowane funkcje.

8. Napisz program zawierający funkcje wykonujące 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:

$$z_1 = 2 + j4 \quad z_2 = 6 - j8$$

Sprawdź poprawność otrzymanych wyników z tabelą 2.

Tabela 1. Poprawne wyniki do zadania 8

Operacja	Wynik	Operacja	Wynik
$z_1 + z_2$	$8 - j4$	$z_1 * z_2$	$44 + j8$
$z_1 - z_2$	$-4 + j12$	z_1 / z_2	$-0,2 + j0,4$

9. Napisz funkcję rekurencyjną obliczającą x^n na podstawie podanego wzoru. Wczytaj x (liczba rzeczywista) i n (liczba naturalna) z klawiatury.

$$x^n = x \cdot x^{n-1} \quad (11)$$

10. Ciąg liczbowy opisany jest podanym wzorem rekurencyjnym:

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

Napisz funkcję rekurencyjną obliczającą n -ty wyraz tego ciągu. Stosując napisaną funkcję oblicz i wyświetl sumę wyrazów tego ciągu 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] Deitel P.J., Deitel H.: Język C. Solidna wiedza w praktyce. Wydanie VIII. Helion, Gliwice, 2020.
- [4] Kochan S.G.: Język C. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2015.
- [5] King K.N.: Język C. Nowoczesne programowanie. Wydanie II. Helion, Gliwice, 2011.

- [6] Reese R.: Wskaźniki w języku C. Przewodnik. Helion, Gliwice, 2014.
- [7] Reek K.A.: Język C. Wskaźniki. Vademecum profesjonalisty. Helion, Gliwice, 2003.
- [8] <http://www.cplusplus.com/reference/clibrary> - C library - C++ Reference
- [9] <https://cpp0x.pl/dokumentacja/standard-C/1> - Standard C
- [10] <https://www.codeblocks.org/> - Code::Blocks

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 argumentów 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.