Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 1

Kod przedmiotu: **EMS1A1004**

(studia stacjonarne)

JĘZYK C - PRZEKAZYWANIE ARGUMENTÓW DO FUNKCJI, REKURENCJA

Numer ćwiczenia

INF1_E11

Autor:
dr inż. Jarosław Forenc

Białystok 2025

Spis treści

Materiały przygotowane w ramach projektu „PB 5.0 - dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji" (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows 10/11.

1.2. Oprogramowanie

Na komputerach zainstalowany jest edytor kodu źródłowego Visual Studio Code 1.103 (lub nowszy) wraz z odpowiednimi rozszerzeniami oraz MinGW - zestaw kompilatorów różnych języków programowania (m.in. C, C++, Fortran).

2. Wiadomości teoretyczne

2.1. 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.

```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

2.2. Przekazywanie argumentów do funkcji przez wartość

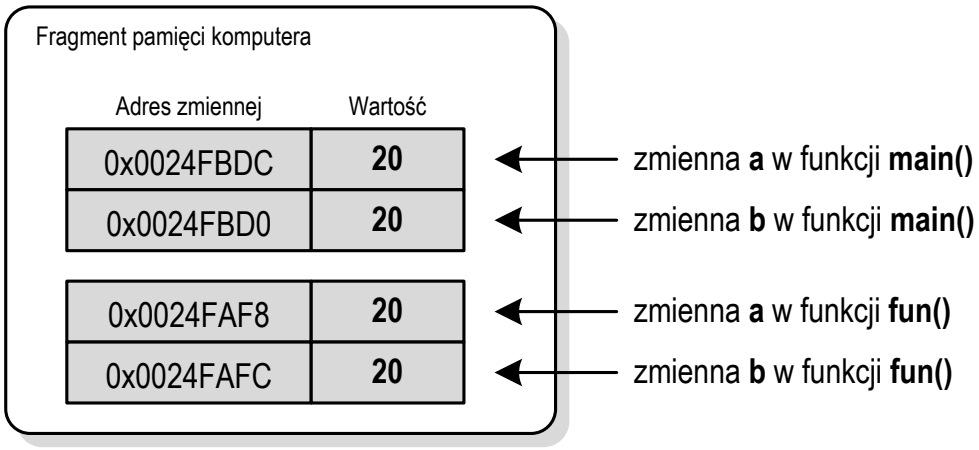
W języku C istnieją dwie metody przekazywania argumentów do funkcji: przez wartość i przez wskaźnik.

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.

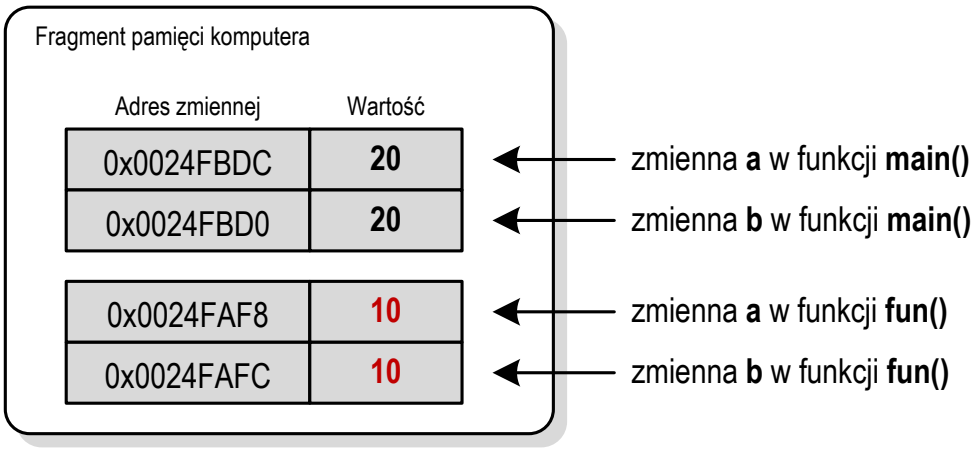
```
#include <stdio.h>

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. 1.



Rys. 1. 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. 2.



Rys. 2. 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
```

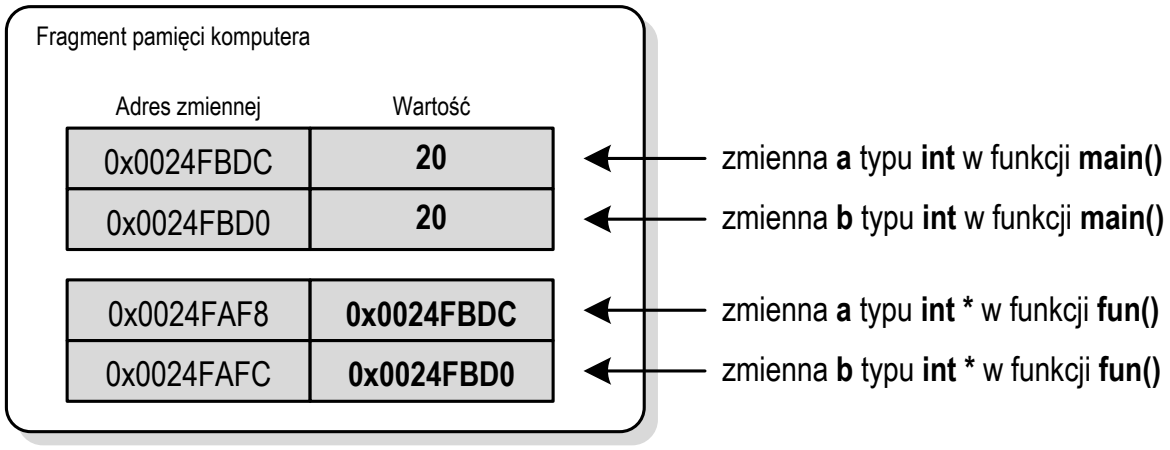
2.3. Przekazywanie argumentów do funkcji przez wskaźnik

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.

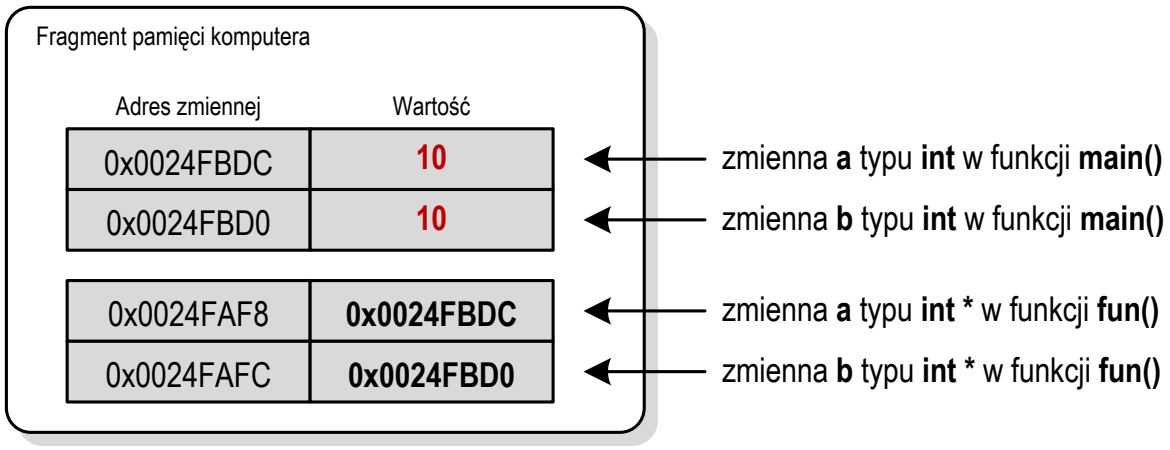
```
#include <stdio.h>

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. 3.



Rys. 3. 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 wyłuskania (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. 4.



Rys. 4. 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 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 = %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 b: -3
Podaj c: 2
x1 = 1
x2 = 2
```

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

```
Podaj a: 2
Podaj b: 2
Podaj c: 1
Brak pierwiastkow
```

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
```

2.4. 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");
}
```

```c
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
```

2.5. 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;
}
```

```c
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
```

2.6. 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 **odległosc()**, 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
```

2.7. 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>=1 ? n*silnia(n-1) : 1;
}
```

- ciąg Fibonacciego:

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

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:

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

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 funkcje wykonujące operacje na **N** - elementowym wektorze liczb całkowitych:

   a) **generuj()** - funkcja zapisująca do wektora wygenerowane pseudolosowo liczby całkowite z zakresu ⟨**a, b**⟩, gdzie **a** i **b** są argumentami funkcji;

   b) **wyswietl()** - funkcja wyświetlająca elementy wektora w jednym wierszu;

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

   d) **ile_x()** - funkcja zwracają liczbę wystąpień wartości **x** w wektorze (**x** - argument funkcji);

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

   Wywołaj w programie wszystkie zdefiniowane funkcje.

2. Napisz program zawierający funkcje wykonujące operacje na **N**×**M** - elementowej macierzy liczb całkowitych:

   f) **generuj()** - funkcja zapisująca do macierzy wygenerowane pseudolosowo liczby całkowite z zakresu ⟨**a, b**⟩, gdzie **a** i **b** są argumentami funkcji;

   g) **wyswietl()** - funkcja wyświetlająca elementy macierzy z podziałem na wiersze i kolumny;

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

   i) **ile_x()** - funkcja zwracają liczbę wystąpień wartości **x** w macierzy (**x** - argument funkcji);

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

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

   Wywołaj w programie wszystkie zdefiniowane funkcje.

3. Tablica liczb rzeczywistych pojedynczej precyzji ma **N** wierszy i **3** kolumny. Pierwsza i druga kolumna zawierają, odpowiednio, współrzędne **x** i **y** punktów opisujących położenie elementów obwodu elektrycznego. Jeden z punktów to źródło napięcia w punkcie odniesienia **(0,0)**. Napisz program, który dla każdego

elementu zapisze w trzeciej kolumnie jego odległość od początku układu współrzędnych. Do obliczenia odległości zastosuj dodatkową funkcję.

4. 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:

z1 = 2 + j4 z2 = 6 - j8

Sprawdź poprawność otrzymanych wyników z tabelą 1.

Tabela 1. Poprawne wyniki do zadania 4

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

5. Wzór rekurencyjny (4) opisuje napięcie w obwodzie RC na kolejnych etapach ładowania kondensatora:

$$V_n = \begin{cases} 5{,}0 & \text{dla} \quad n = 0 \\ 2{,}0 & \text{dla} \quad n = 1 \\ 0{,}8 \cdot V_{n-1} + 0{,}15 \cdot V_{n-2} & \text{dla} \quad n > 1 \end{cases} \tag{4}$$

Napisz funkcję rekurencyjną, która oblicza n-ty wyraz tego ciągu. Stosując napisaną funkcję oblicz i wyświetl sumę napięć od V_0 do V_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] Stańczyk J.: Nowoczesny C: przegląd C23 z przykładami. Helion, Gliwice, 2023.

[7] http://www.cplusplus.com/reference/clibrary - C library - C++ Reference

[8] https://cpp0x.pl/dokumentacja/standard-C/1 - Standard C

[9] https://code.visualstudio.com/ - Visual Studio Code

5. Pytania kontrolne

1. Wyjaśnij różnice w przekazywaniu argumentów do funkcji przez wartość i wskaźnik.
2. Opisz sposób przekazywania do funkcji tablic jedno- i dwuwymiarowych oraz struktur.
3. 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.