Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 2

Kod przedmiotu: **ES1D300 017**

(studia stacjonarne)

JĘZYK C - WSKAŹNIKI, DYNAMICZNY PRZYDZIAŁ PAMIĘCI

Numer ćwiczenia

INF24

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.

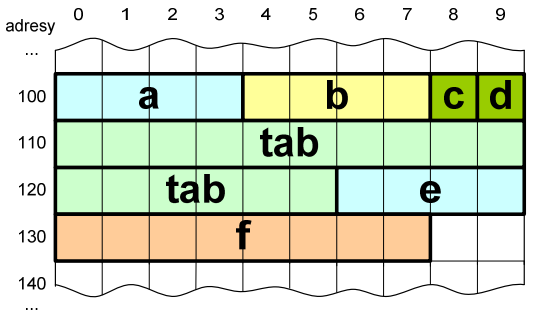
2. Wiadomości teoretyczne

2.1. Wskaźniki

Wskaźnik jest **zmienną** mogącą zawierać adres obszaru pamięci, a w szczególności adres innej zmiennej (obiektu). Załóżmy, że mamy następujące deklaracje zmiennych:

```
int a;
float b;
char c, d;
int tab[4];
int e;
double f;
```

Wszystkie zmienne przechowywane są w pamięci komputera. Zależnie od typu zmiennej zajmują określoną liczbę bajtów (Rys. 1). Każda zmienna oprócz nazwy ma także adres. Program nie posługuje się nazwami zmiennych tylko ich adresami. Na przykład zmienna **a** typu **int** zajmuje 4 bajty i znajduje się pod adresem **100**, tablica **tab** przechowująca cztery elementy typu **int**, znajduje się pod adresem **110** i zajmuje 4 × 4 = 16 bajtów.



Rys. 1. Zmienne w pamięci komputera

Adres określa miejsce w pamięci, natomiast nie informuje o rozmiarze wskazywanego obiektu. Deklarując wskaźnik (zmienną wskazującą) musimy podać typ obiektu, na jaki on wskazuje. Deklaracja zmiennej wskazującej wygląda tak samo, jak deklaracja każdej innej zmiennej, tylko że jej nazwa poprzedzona jest symbolem gwiazdki (*):

typ *nazwa_zmiennej;

lub

typ* nazwa_zmiennej;

lub

typ * nazwa_zmiennej;

Poniższy zapis zawiera deklarację zmiennej wskaźnikowej do typu **int**. Oznacza to, że zmienna **ptr** jest typu: **wskaźnik do zmiennej typu int**. Zmienna **ptr** może przechowywać adresy zmiennych **a** i **e** z wcześniejszego przykładu.

```
int *ptr;
```

Do przechowywania adresu zmiennej typu **double** należy zadeklarować zmienną typu: **wskaźnik do zmiennej typu double**.

```
double *ptrd;
```

Można konstruować wskaźniki do danych dowolnego typu łącznie z typami **wskaźnik do...** lub **wskaźnik do wskaźnika do...** .

```
char **wsk;
```

W powyższym przykładzie **wsk** jest typu **wskaźnik do wskaźnika do typu char**. Można deklarować tablice wskaźników. Zmienna **tab_wsk** jest tablicą zawierającą 10 wskaźników do typu **int**.

```
int *tab_wsk[10];
```

Deklarując wskaźniki lepiej jest pisać * przy zmiennej, a nie przy typie:

```
int *ptr1;    /* lepiej */
int* ptr2;    /* gorzej */
```

gdyż trudniej jest pomylić się przy deklaracji dwóch wskaźników:

```
int *p1, *p2;
int* p3, p4;
```

Zmienne **p1**, **p2** i **p3** są wskaźnikami do typu **int**, zaś zmienna **p4** jest „zwykłą” zmienną typu **int**.

Do przypisania wskaźnikowi wartości (czyli adresu) można zastosować jednoargumentowy operator pobierania adresu **&**:

```
int a = 10;    /* a - zmienna typu int */
int *ptr;      /* ptr - wskaźnik do typu int */
ptr = &a;
```

Poprzez adres zmiennej można „dostać się” do jej wartości używając tzw. **operatora wyłuskania** (**odwołania pośredniego**) - gwiazdki (*):


```
*ptr = 20;
```

Jeśli zmienna **ptr** przechowuje adres zmiennej **a**, to powyższe przypisanie jest równoważne nadaniu zmiennej **a** wartości **20**:

```
a = 20;
```

Wskaźnik pusty to specjalna wartość, odróżnialna od wszystkich innych wartości wskaźnikowych, dla której gwarantuje się nierówność ze wskaźnikiem do dowolnego obiektu. Do zapisu wskaźnika pustego stosuje się wyrażenie całkowite o wartości zero (**0**):

```
int *ptr = 0;
```

Często zamiast wartości **0** stosowana jest makrodefinicja preprocesora **NULL**, która podczas kompilacji programu zamieniana jest na **0**:

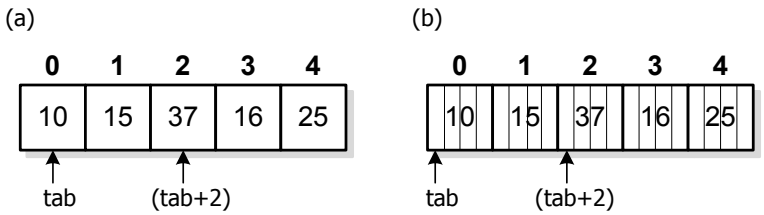
```
int *ptr = NULL;
```

2.2. Związek tablic ze wskaźnikami

W języku C istnieje ścisły związek tablic ze wskaźnikami. Załóżmy, że dana jest następująca deklaracja tablicy i jej inicjalizacja:

```
int tab[5] = {10,15,37,16,25};
```

Nazwa tablicy jest adresem zerowego elementu tablicy (Rys. 2a). Dokładniej mówiąc - adresem pierwszego bajtu zajmowanego przez ten element (Rys. 2b).



Rys. 2. Odwołania do elementów tablicy

Jeśli nazwa tablicy jest adresem zerowego jej elementu, to stawiając przed nią symbol * możemy „dostać się" do wartości tego elementu. Zatem:

`*tab`	jest równoważne:	`tab[0]`

Podobnie:

`*(tab+1)`
`*(tab+2)`
`...`
	jest równoważne:	`tab[1]`
		`tab[2]`
		`...`

oraz ogólnie:

`*(tab+i)`	jest równoważne:	`tab[i]`

W zapisie ***(tab+i)** nawiasy są konieczne, gdyż operator * ma wyższy priorytet niż operator **+**. Poniższy przykład pokazuje różnice w zapisie z nawiasami i bez nawiasów.

```
int tab[5] = {10,15,37,16,25}, x;
x = *(tab+2);
printf("x = %d",x);            /*  x = 37 */
x = *tab+2;
printf("x = %d",x);            /*  x = 12 */
```


W pierwszym przypadku zmiennej **x** przypisywana jest wartość **37**, gdyż taką wartość ma element tablicy **tab** o indeksie **2** (zapis: **x = *(tab+2)** jest równoważny zapisowi: **x = tab[2]**). W drugim przypadku zmiennej **x** zostanie przypisana suma elementu tablicy **tab** o indeksie **0** i liczby **2** (zapis: **x = *tab+2** jest równoważny zapisowi: **x = tab[0] + 2**).

2.3. Pamięć a zmienne w programie

Ze względu na czas życia wyróżnia się w programie w języku C:

- **obiekty statyczne** - istnieją od chwili rozpoczęcia działania programu aż do jego zakończenia;
- obiekty **dynamiczne** - tworzone i usuwane z pamięci w trakcie wykonania programu - automatycznie (bez udziału programisty) lub kontrolowane przez programistę.

O typie obiektu (statyczny lub dynamiczny) decyduje **miejsce deklaracji** obiektu w kodzie programu oraz **klasa pamięci** obiektu. Ze względu na miejsce deklaracji obiektu w kodzie programu wyróżnia się:

- zmienne **globalne** (deklarowane poza funkcjami), które są statyczne;
- zmienne **lokalne** (deklarowane wewnątrz bloków funkcyjnych), których zaliczenie do obiektów statycznych lub dynamicznych zależy od klasy pamięci.

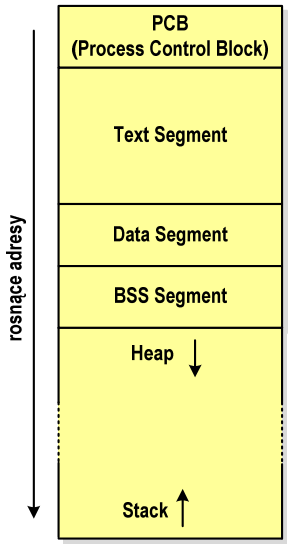
Klasę pamięci określają słowa kluczowe języka C:

- **auto** - zmienna automatyczna;
- **static** - zmienna statyczna;
- **register** - zmienna umieszczana w rejestrach procesora (automatyczna).

```
int x;              /* zmienna automatyczna */
static int y;       /* zmienna statyczna    */
register int z;     /* zmienna automatyczna */
```

Wszystkie zmienne lokalne zadeklarowane bez jawnego specyfikowania klasy pamięci są automatyczne (**auto**).

Kiedy uruchamiany jest program komputerowy to jest on ładowany do pamięci operacyjnej komputera. Struktura programu w pamięci przedstawiona jest na poniższym rysunku.



Rys. 3. Struktura programu w pamięci komputera

Blok kontrolny procesu (*PCB - Process Control Block*) jest obszarem pamięci operacyjnej zarezerwowanym przez system operacyjny do zarządzania procesem. **Segment kodu** (*Text Segment*) zawiera kod programu czyli instrukcje w postaci binarnej. **Segment danych** składa się z dwóch części. W pierwszej części (*Data Segment*) umieszczane są zmienne globalne i statyczne zainicjalizowane niezerowymi wartościami. W drugiej części (*BSS Segment - Block Started by Symbol*) znajdują się także zmienne globalne i statyczne, ale domyślnie zainicjalizowane zerowymi wartościami. **Stos** (*Stack*) przechowuje zmienne lokalne (automatyczne) oraz parametry i adresy powrotu z funkcji - ramki stosu (*Stack Frame*). **Sterta** (*Heap*) jest obszarem zmiennych dynamicznych. Stos programu jest ograniczony co do rozmiaru. Jeżeli program wymaga zadeklarowania bardzo dużej tablicy, to nie wykonuje się tego na stosie, ale na stercie. W takim przypadku należy zastosować dynamiczny przydział pamięci.

2.4. Dynamiczny przydział pamięci w języku C

Do dynamicznego przydziału pamięci w języku C stosowane są dwie funkcje: **calloc()** i **malloc()**. Przydział bloków pamięci następuje w obszarze sterty (stosu zmiennych dynamicznych). Przydzieloną dynamicznie pamięć należy zwolnić wywołując funkcję **free()**. Zastosowanie wymienionych funkcji wymaga dołączenia pliku nagłówkowego **stdlib.h**.

`calloc()`	Nagłówek: void ***calloc(size_t n, size_t size);**

- funkcja **calloc()** przydziela blok pamięci o rozmiarze **n*size** (mogący pomieścić tablicę **n**-elementów rozmiaru **size** każdy) i zwraca wskaźnik do tego bloku;
- jeśli nie można przydzielić pamięci, to funkcja zwraca wartość **NULL**;
- przydzielona pamięć jest inicjowana zerami (bitowo).

`malloc()`	Nagłówek: void ***malloc(size_t size);**

- funkcja **malloc()** przydziela blok pamięci o rozmiarze określonym parametrem **size** i zwraca wskaźnik do tego bloku;
- jeśli pamięci nie można przydzielić, to funkcja zwraca wartość **NULL**;
- przydzielona pamięć nie jest inicjowana.

Jeśli przydzielony blok pamięci nie jest już potrzebny, to należy zwolnić pamięć wywołując funkcję **free()**.

`free()`	Nagłówek: void ***free(**void ***ptr);**

- funkcja **free()** zwalnia blok pamięci wskazywany parametrem **ptr**;
- wartość **ptr** musi być wynikiem wywołania funkcji **calloc()** lub **malloc()**.

W poniższym programie przydzielana jest dynamicznie pamięć na tablicę liczb całkowitych, której rozmiar wprowadza użytkownik z klawiatury. Do tablicy zapisywane są liczby podawane przez użytkownika oraz obliczana jest ich średnia arytmetyczna.

Dynamiczny przydział pamięci na tablicę jednowymiarową.

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

int main(void)
{
    int   *tab, i, n, x;
    float suma = 0.0;

    printf("Podaj ilosc liczb: ");
    scanf("%d",&n);

    tab = (int *) calloc(n,sizeof(int));
    if (tab == NULL)
    {
        printf("Nie mozna przydzielic pamieci.\n");
        return -1;
    }

    for (i=0; i<n; i++)   /* wczytanie liczb */
    {
        printf("Podaj liczbe nr %d: ",i+1);
        scanf("%d",&x);
        tab[i] = x;
    }

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

    printf("Srednia %d liczb wynosi %f\n",n,suma/n);

    free(tab);

    return 0;
}
```

Do przydziału pamięci na tablicę zastosowano funkcję **calloc()**. Funkcja ta zwraca wskaźnik do typu **void**, dlatego wartość wskaźnika należy rzutować na właściwy

typ (**int ***). Jeśli nie było możliwe przydzielenie pamięci (**tab == NULL**), to wyświetlany jest odpowiedni komunikat i program kończy pracę. Po przydzieleniu pamięci następuje wczytanie liczb do tablicy (pierwsza pętla **for**). Sposób odwoływania się do elementów tablicy jest identyczny, jak w przypadku zwykłej tablicy (**tab[i]**). Liczby można wczytywać bezpośrednio do tablicy **tab**, co pokazuje poniższy kod:

```
for (i=0; i<n; i++)   /* wczytanie liczb */
{
    printf("Podaj liczbe nr %d: ",i+1);
    scanf("%d",&tab[i]);
}
```

Wczytywanie liczb można zapisać jeszcze w inny sposób:

```
for (i=0; i<n; i++)   /* wczytanie liczb */
{
    printf("Podaj liczbe nr %d: ",i+1);
    scanf("%d",(tab+i));
}
```

Po wczytaniu liczb obliczana jest ich suma (druga pętla **for**), a następnie średnia arytmetyczna. Na koniec funkcja **free()** zwalnia pamięć przydzieloną na tablicę. Zamiast funkcji **calloc()** można zastosować funkcję **malloc()**:

```
tab = (int *) malloc(n*sizeof(int));
```

2.5. Dynamiczny przydział pamięci na macierz

Przedstawione w poprzednim rozdziale funkcje **malloc()** i **calloc()** umożliwiają dynamiczny przydział pamięci tylko na wektor elementów. W przypadku macierzy należy zastosować inny sposób. Trzy metody przydziału pamięci na macierz zawierającą **N** wierszy i **M** kolumn przedstawiono poniżej.

Metoda 1 (wektor N×M - elementowy)

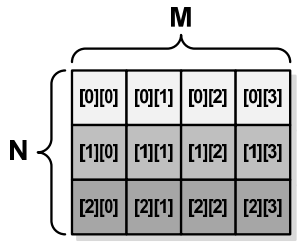
Elementy macierzy umieszczone są wierszami w wektorze N×M - elementowym. Przydzielamy pamięć na wektor:

```
int *tab;
tab = (int *) calloc(N*M,sizeof(int));
```

Po wykorzystaniu wektora, przydzieloną pamięć zwalniamy funkcją **free()**:

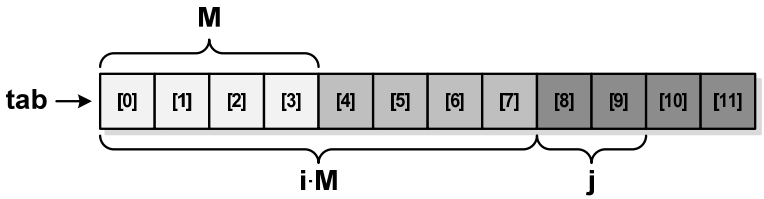
```
free(tab);
```

Niestety przy odwoływaniu się do elementów takiej macierzy nie możemy używać standardowego indeksowania **tab[i][j]**, gdyż **tab** jest w rzeczywistości wektorem elementów typu **int**. Należy przeliczyć indeks wiersza (**i**) oraz indeks kolumny (**j**) na indeks odpowiedniego elementu wektora. Załóżmy, że macierz ma **N = 3** wiersze i **M = 4** kolumny (Rys. 4).



Rys. 4. Macierz mająca N wierszy i M kolumn

Macierz w postaci wektora N×M - elementowego przedstawia Rys. 5.



Rys. 5. Macierz w postaci wektora N×M - elementowego

Odwołanie do **i**-tego wiersza i **j**-tej kolumny macierzy ma postać:

```
tab[i*M+j]
```

lub

```
*(tab+i*M+j)
```

W poniższym programie pokazano dynamiczny przydział pamięci na macierz. Zapisywane są do niej wygenerowane pseudolosowo liczby całkowite. Następnie elementy macierzy wyświetlane są na ekranie z podziałem na wiersze i kolumny. Na koniec pamięć zajmowana przez macierz jest zwalniana.

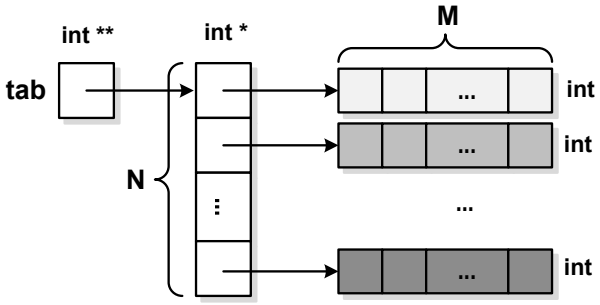
Przydział pamięci na macierz w postaci wektora N×M - elementowego.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 4   /* liczba wierszy */
#define M 6   /* liczba kolumn */

int main(void)
{
    int i, j, *tab;

    tab = (int *) calloc(N*M,sizeof(int));

    srand((unsigned int)time(NULL));

    for (i=0; i<N; i++)
        for (j=0; j<M; j++)
            tab[i*M+j] = rand()%100;

    for (i=0; i<N; i++)
    {
        for (j=0; j<M; j++)
            printf("%4d",tab[i*M+j]);
        printf("\n");
    }

    free(tab);

    return 0;
}
```

Metoda 2 (wskaźnik na tablicę wskaźników)

Przydzielamy pamięć na **N**-elementowy wektor wskaźników do typu **int**. Następnie przydzielamy pamięć na **N** wektorów **M**-elementowych zawierających elementy typu **int**, zapisując jednocześnie ich adresy do kolejnych elementów wektora wskaźników (Rys. 6).

```
int **tab;
tab = (int **) calloc(N,sizeof(int *));
for (i=0; i<N; i++)
    tab[i] = (int *) calloc(M,sizeof(int));
```



Rys. 6. Implementacja macierzy w postaci wskaźnika do tablicy wskaźników

Po wykorzystaniu tablicy należy zwolnić przydzieloną pamięć. W pierwszej kolejności zwalniamy pamięć przydzieloną na **N** wektorów, każdy o rozmiarze **M**, a następnie zwalniamy pamięć przydzieloną na **N**-elementowy wektor wskaźników do typu **int**.

```
for (i=0; i<N; i++)
    free(tab[i]);
free(tab);
```

Odwołania do elementów mają taką samą postać, jak w przypadku „zwykłych" tablic: **tab[i][j]**. Przykład zastosowania przydziału pamięci na macierz w postaci wskaźnika do tablicy wskaźników przedstawia poniższy program.

Przydział pamięci na macierz w postaci wskaźnika do tablicy wskaźników.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 4    /* liczba wierszy */
#define M 6    /* liczba kolumn */

int main(void)
{
    int i, j, **tab;

    tab = (int **) calloc(N,sizeof(int *));
    for (i=0;i<N;i++)
        tab[i] = (int *) calloc(M,sizeof(int));

    srand((unsigned int)time(NULL));

    for (i=0; i<N; i++)
        for (j=0; j<M; j++)
            tab[i][j] = rand()%100;

    for (i=0; i<N; i++)
    {
        for (j=0; j<M; j++)
            printf("%4d",tab[i][j]);
        printf("\n");
    }

    for (i=0; i<N; i++)
        free(tab[i]);
    free(tab);

    return 0;
}
```

Pamięć przydzielona przy zastosowaniu tej metody nie stanowi ciągłego obszaru w pamięci komputera. W pewnych sytuacjach może stanowić to przeszkodę przy opracowywaniu wydajnych algorytmów pracujących na takiej macierzy. Problemu ciągłości obszaru pamięci zajmowanej przez macierz nie ma w przypadku trzeciej metody.

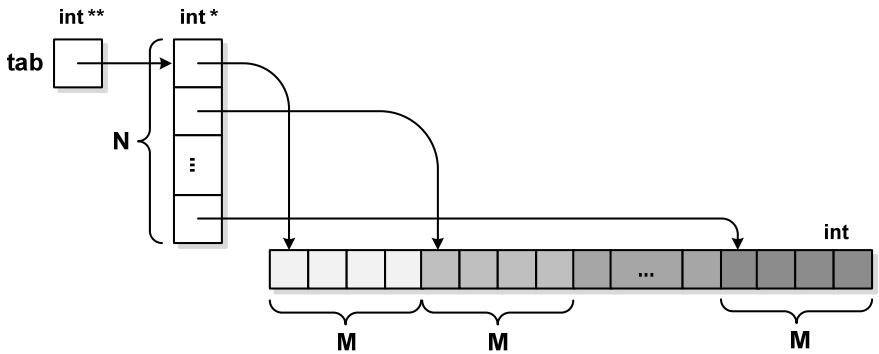
Metoda 3 (wektor N×M - elementowy + wektor wskaźników)

Przydzielamy pamięć na **N** - elementowy wektor wskaźników do typu **int**. Następnie przydzielamy pamięć na **N×M** - elementowy wektor elementów typu **int**, którego adres zapisujemy pod zerowym elementem wektora **N** - elementowego. Do pozostałych elementów wektora **N** - elementowego zapisujemy adresy pierwszych elementów kolejnych wierszy macierzy znajdujących się w wektorze **N×M** - elementowym (Rys. 7).

```
int **tab;
tab = (int **) calloc(N,sizeof(int *));
tab[0] = (int *) calloc(N*M,sizeof(int));
for (i=1; i<N; i++)
    tab[i] = tab[0]+i*M;
```

Po wykorzystaniu tablicy należy zwolnić przydzieloną pamięć. W pierwszej kolejności zwalniamy pamięć przydzieloną na **N×M** - elementowy wektor elementów typu **int**, a następnie na **N** - elementowy wektor wskaźników.

```
free(tab[0]);
free(tab);
```



Rys. 7. Implementacja macierzy w postaci wektora N×M - elementowego i wektora wskaźników

Odwołania do elementów mają taką samą postać, jak w przypadku „zwykłych" tablic: **tab[i][j]**. Przykład zastosowania przydziału pamięci na macierz w postaci wektora **N×M** - elementowego i wektora wskaźników przedstawia poniższy program.

Przydział pamięci na macierz w postaci wektora N×M - elementowego i wektora wskaźników.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 4   /* liczba wierszy */
#define M 6   /* liczba kolumn */

int main(void)
{
    int i, j, **tab;

    tab = (int **) calloc(N,sizeof(int *));
    tab[0] = (int *) calloc(N*M,sizeof(int));
    for (i=1; i<N; i++)
        tab[i] = tab[0]+i*M;

    srand((unsigned int)time(NULL));

    for (i=0; i<N; i++)
        for (j=0; j<M; j++)
            tab[i][j] = rand()%100;

    for (i=0; i<N; i++)
    {
        for (j=0; j<M; j++)
            printf("%4d",tab[i][j]);
        printf("\n");
    }

    free(tab[0]);
    free(tab);

    return 0;
}
```

3. Przebieg ćwiczenia

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

1. Zadeklaruj w programie zmienne typów: **float**, **char**, **double**, **int**, tablicę 10 elementów typu **char**, zmienną typu **short**. Wyświetl adresy wszystkich zadeklarowanych zmiennych (zastosuj specyfikator formatu **%p**). Na podstawie adresów określ:
   - czy zmienne znajdują się w pamięci komputera w takiej samej kolejności, jak w deklaracji?
   - czy zmienne znajdują się w pamięci komputera bezpośrednio po sobie?

2. Napisz program, w którym użytkownik wprowadza z klawiatury liczbę elementów wektora. Przydziel dynamicznie pamięć na wektor liczb całkowitych. Wykonaj następujące operacje:
   - zapisz do wektora wygenerowane pseudolosowo liczby całkowite z przedziału ⟨**0, 99**⟩;
   - wyświetl na ekranie elementy wektora;
   - oblicz i wyświetl sumę oraz średnią arytmetyczną elementów wektora;
   - znajdź i wyświetl wartość elementu o największej wartości.

3. Napisz program, w którym użytkownik wprowadza z klawiatury liczbę wierszy i liczbę kolumn macierzy. Przydziel dynamicznie pamięć na macierz liczb całkowitych. Wykonaj następujące operacje:
   - zapisz do macierzy wygenerowane pseudolosowo liczby całkowite z przedziału ⟨**-99, 99**⟩;
   - wyświetl na ekranie elementy macierzy z podziałem na wiersze i kolumny;
   - oblicz i wyświetl sumę oraz średnią arytmetyczną elementów macierzy;
   - znajdź i wyświetl wartość elementu o największej wartości.

4. Napisz program wyświetlający argumenty funkcji **main** (każdy argument w oddzielnym wierszu).
5. Napisz program obliczający i wyświetlający sumę liczb całkowitych przekazanych do programu jako argumenty funkcji **main**.

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.

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

5. Pytania kontrolne

1. Wyjaśnij pojęcie wskaźnika, podaj jak deklaruje się wskaźniki.
2. Jaki jest związek tablic ze wskaźnikami w języku C?
3. Scharakteryzuj obiekty statyczne i dynamiczne występujące w kodzie programu.
4. Opisz funkcje do dynamicznego przydzielania i zwalniania pamięci w języku C.
5. Opisz wybraną metodę przydziału pamięci dla macierzy.

6. Literatura

[9] Prata S.: Język C. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2016.

[10] Kernighan B.W., Ritchie D.M.: Język ANSI C. Programowanie. Wydanie II. Helion, Gliwice, 2010.

[11] Prinz P., Crawford T.: Język C w pigułce. APN Promise, Warszawa, 2016.

[12] King K.N.: Język C. Nowoczesne programowanie. Wydanie II. Helion, Gliwice, 2011.

[13] Kochan S.G.: Język C. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2015.

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

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

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

7. Zagadnienia na zaliczenie

6. Wyjaśnij pojęcie wskaźnika, podaj jak deklaruje się wskaźniki.
7. Jaki jest związek tablic ze wskaźnikami w języku C?
8. Scharakteryzuj obiekty statyczne i dynamiczne występujące w kodzie programu.
9. Opisz funkcje do dynamicznego przydzielania i zwalniania pamięci w języku C.
10. Opisz wybraną metodę przydziału pamięci dla macierzy.

8. 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.