

Lab 8. Tablice liczbowe cd. Operacje na tablicach o dwóch indeksach. Dynamiczna alokacja pamięci dla tablic wielowymiarowych. Projekt nr 3.

1. Wektory i macierze:

- a. Przykład 1. Przechowywanie macierzy w tablicy jednowymiarowej „wiersz po wierszu”. i – numer wiersza, j – numer kolumny, kolejność indeksowania:

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        x[n * i + j] = a[i][j];
```

$$\begin{array}{l} \text{Indeks ij} \\ \text{ij} = n \cdot i + j \end{array} \longrightarrow \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

- b. Przykład 2. Przechowywanie macierzy w tablicy jednowymiarowej „kolumna po kolumnie”. i – numer wiersza, j – numer kolumny, kolejność indeksowania:

```
for (j = 0; j < n; j++)
    for (i = 0; i < n; i++)
        x[n * j + i] = a[i][j];
```

$$\begin{array}{l} \text{Indeks ij} \\ \text{ij} = n \cdot j + i \end{array} \longrightarrow \begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

- c. Przykład 3. Mnożenie macierzy przez macierz $c_{ij} = \sum_k a_{ik} \cdot b_{kj}$ metodą:

- a) klasyczną, czyli „wiersz po wierszu”:

```
for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
        for (k = 0; k < K; k++)
            Cij = Cij + Aik * Bkj
```

Element macierzy C w pętli wewnętrznej nie zależy od indeksu k. Pobieranie elementów macierzy A następuje ciągle, bez skoków, natomiast elementy macierzy B pobierane są ze skokami. **W związku z tym metoda ta nie jest skuteczna.**

- b) Przypieszoną:

```
for (i = 0; i < M; i++)
    for (k = 0; k < K; k++)
        for (j = 0; j < N; j++)
            Cij = Cij + Aik * Bkj
```

Elementy macierzy A w pętli wewnętrznej nie zależą od indeksu j. Pobieranie elementów macierzy C następuje ciągle, bez skoków, natomiast elementy macierzy B pobierane są ze skokami. **W związku z tym metoda ta jest lepsza od poprzedniej.**

2. **Dynamiczna alokacja pamięci dla tablic wielowymiarowych** - Przykładowa tablica 10x5 o elementach typu double.

Wersja 1.

```
double **tab;
int n = 10, m = 5;

tab = (double **)malloc(sizeof(double *) * n);
if (!tab)
{
    //błąd alokacji
}
for (int i = 0; i < n; i++)
{
    tab[i] = (double *)malloc(sizeof(double) * m);
    if (!tab[i])
    {
        //błąd alokacji
    }
}
if (tab){
    for (int i = 0; i < n; i++)
    {
        if (tab[i])
        {
            free(tab[i]);
        }
    }
    free(tab);
    tab = NULL;
}
```

Wersja 2.

```
double **tab_a, *tab_b;
int n = 10, m = 5;

tab_b = (double *)malloc(sizeof(double) * n * m );
if (!tab_b)
{
    //błąd alokacji
}
tab_a = (double **)malloc(sizeof(double *) * n);
if (!tab_a)
{
    //błąd alokacji
}

for (int i = 0; i < n; i++)
{
    tab_a[i] = &tab_b[ m * i ];
}

if (tab_b){
    free(tab_b); // free(tab_a[0]);
    tab_b = NULL;
}

if (tab_a){
    free(tab_a);
    tab_a = NULL;
}
```

3. Przykład. Napisz program, który dla danych dwóch macierzy:
 $\bar{A} = \{a_{ij}\}, i = 0,1,2, \dots, n-1; j = 0,1,2, \dots, m-1,$
 $\bar{B} = \{b_{ij}\}, i = 0,1,2, \dots, n-1; j = 0,1,2, \dots, m-1$
 oraz wektora $\vec{x} = \{x_i\}, i = 0,1,2, \dots, m-1$, policzy:
 a) Sumę macierzy: $\bar{C} = \bar{A} + \bar{B}$, czyli $c_{ij} = a_{ij} + b_{ij};$
 b) Iloczyn macierzy \bar{A}_{ij} przez wektor \vec{x} : $\vec{y} = \bar{A}\vec{x}$; $y_i = \sum_{k=0}^{m-1} a_{ik} \cdot x_k; i = 0,1,2, \dots, n-1$.
 Do kontroli poprawności obliczeń wykorzystaj funkcję error z pliku util_1.cpp (patrz lab. 7). Dane pobrać z pliku, wyniki zapisać do pliku, nazwy plików przekazać przez argumenty wywołania programu.

```
//=====PLIK_1=====
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#pragma warning(disable:4996)
#define LL 20
FILE *fw, *fd;
void argumenty(int, char **);
extern void error(int, char *);

int main(int argc, char *argv[])
{
    double a[LL][LL], b[LL][LL], c[LL][LL], x[LL], y[LL], r;
    int i, j, k, n, m;

    argumenty(argc, argv);
    if (!(fd = fopen(argv[1], "r"))) error(2, "dane");
    if (!(fw = fopen(argv[2], "w"))) error(2, "wyniki");

    fscanf(fd, "%d %d", &n, &m);

    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            fscanf(fd, "%lf", &a[i][j]);

    for (i = 0; i < m; i++)
        fscanf(fd, "%lf", &x[i]);

    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            fscanf(fd, "%lf", &b[i][j]);

    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            c[i][j] = a[i][j] + b[i][j];

    for (i = 0; i < n; i++)
    {
        r = 0;
        for (k = 0; k < m; k++)
            r += a[i][k] * x[k];
        y[i] = r;
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            fprintf(fw, "%lf ", c[i][j]);
        fprintf(fw, "\n");
    }
}
```

```

        for (i = 0; i < n; i++)
        {
            fprintf(fw, "%lf ", y[i]);
            if (!((i + 1) % 5)) fprintf(fw, "\n");
        }
        system("pause");
        return 0;
    }

    void argumenty(int argc, char *argv[])
    {
        int len;
        char *usage;
        if (argc != 3)
        {
            len = strlen(argv[0]) + 19;
            if (!(usage = (char*)malloc((unsigned)len * sizeof(char))))
                error(3, "tablica usage");
            strcpy(usage, argv[0]);
            strcat(usage, " file_in file_out");
            error(4, usage);
        }
    }

/************* plik util_1.cpp *****/
#include<stdio.h>
#define MAX_ERR 5
static char *p[] = { "",  

                    " zle dane",  

                    " otwarcie pliku",  

                    " brak pamieci",  

                    " Usage : ",  

                    " nieznany ",  

};

void error(int nr, char *str)
{
    int k;
    k = nr >= MAX_ERR ? MAX_ERR : nr;
    fprintf(stderr, "Blad(%d) - %s %s\n", nr, p[k], str);
    system("pause");
    exit(nr);
}

```

4. Zmodyfikuj program z powyższego przykładu tak aby:

- Zamiast tablic jednowymiarowych zdefiniować zmienne typu `double *`, zamiast tablic dwuwymiarowych zdefiniować zmienne typu `double **`;
- Zarezerwować dokładnie m miejsc na tablice jednowymiarowe typu `double` (alokacja pamięci), np.:


```
x = (double*)malloc((unsigned)m * sizeof(double));
```
- Zarezerwować dokładnie n*m miejsc na tablice dwuwymiarowe typu `double` (alokacja pamięci patrz pkt.2), np.:


```
a = (double**)malloc((unsigned)n * sizeof(double*));
for (i = 0; i < n; i++)
    a[i] = (double*)malloc((unsigned)m * sizeof(double));
```

5. Algorytm z pkt.3 podziel na kilka funkcji:

- a) Alokacja pamięci (dwie funkcje - wer.1 i wer.2 – patrz pkt.2) :

```
double **DajMac_1(int n, int m)
double **DajMac_2(int n, int m)
```

b) Zwolnienie pamięci (dwie funkcje):

```
void ZwrocMac_1(double **ma, int n, int m)
void ZwrocMac_2(double **ma, int n, int m)
```

c) Czytanie elementów tablicy o dwóch indeksach z pliku:

```
void CzytMac(FILE *fd, double **ma, int n, int m)
{
    int i, j;
    char *err;
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
    {
        if (fscanf(fd, "%lf", &ma[i][j]) != 1)
        {
            printf("blad - element nr %d %d\n", i, j);
        }
    }
}
```

d) Pisanie elementów tablicy o dwóch indeksach do pliku:

```
void PiszMak(FILE *fw, double **ma, int n, int m)
```

e) Dodawanie macierzy:

```
void DodMac(double **ma1, double **ma2, double **ma3, int n, int m)
```

f) Mnożenie macierzy przez wektor:

```
void Mac_x_Wekt(double **ma, double *we, double *wy, int n, int m)
```

g) Napisz dodatkowo funkcję, która dla danych macierzy:

$$\bar{X} = \{x_{ij}\}, i = 0, 1, 2, \dots, n-1; j = 0, 1, 2, \dots, m-1,$$

$$\bar{Y} = \{y_{ij}\}, i = 0, 1, 2, \dots, m-1; j = 0, 1, 2, \dots, p-1$$

policzy macierz $\bar{Z} = \bar{X}\bar{Y}$ czyli

$$z_{ij} = \sum_{k=0}^{m-1} x_{ik} y_{kj}, \quad i = 0, 1, 2, \dots, n-1, \quad j = 0, 1, 2, \dots, p-1$$

```
void Mac_x_Mac(double **x, double **y, double **z, int n, int m, int p)
```

Funkcje realizujące zadania a), b), c), d) zapisz w pliku util_4.cpp.

Funkcje realizujące zadania e), f), g) zapisz w pliku util_5.cpp.

6. Zrealizuj zadania z powyższego przykładu (pkt.2 i następne) wykorzystując utworzone wcześniej funkcje zawarte w plikach util_2.cpp, util_3.cpp (lab. 7), util_4.cpp, util_5.cpp.

Fragment programu:

```
FILE *fw, *fd;
extern void error(int, char*), PiszMak(FILE *, double *, int),
CzytWekt(FILE *, double *, int), CzytMac(FILE *, double **, int, int),
DodMac(FILE *, double **, int, int);
extern double *DajWekt(int), **DajMac_1(int, int);
extern void Mac_x_Wekt(double **, double *, double *, int, int);
void argumenty(int argc, char *argv[]);
int main(int argc, char *argv[])
{
    double *x, *y;
    double **a, **b, **c;
    int n, m;

    argumenty(argc, argv);
    if (!(fd = fopen(argv[1], "r")))) error(2, "dane");
    if (!(fw = fopen(argv[2], "w")))) error(2, "wyniki");
    fscanf(fd, "%d %d", &n, &m);
    x = DajWekt(m);
```

```

y = DajWekt(m);
a = DajMac_1(n, m);
b = DajMac_1(n, m);
c = DajMac_1(n, m);
CzytMac(fd, a, n, m);
CzytMac(fd, b, n, m);
CzytWekt(fd, x, m);
DodMac(a, b, c, n, m);
Mac_x_Wekt(a, x, y, n, m);
printf("Macierz\n");
PiszMac(stdout, c, n, m);
fprintf(fw, "Macierz\n");
PiszMac(fw, c, n, m);
printf("Wektor\n");
PiszWekt(stdout, y, n);
fprintf(fw, "Wektor\n");
PiszWekt(fw, y, n);
}

```

7. Dana jest macierz kwadratowa $B[n][n]$. Znaleźć sumę elementów leżących poniżej głównej przekątnej i jednocześnie należących do zadanego przedziału $[a,b]$.
8. Dana jest macierz kwadratowa $B[n][n]$. Znaleźć sumę tych elementów leżących na obu przekątnych macierzy, które jednocześnie spełniają warunek (na głównej przekątnej $j=i$, na drugiej $j=n-i$):

$$\sin(B[i][j]) \geq \begin{cases} 0 & \text{dla elementów na głównej przekątnej} \\ 0,5 & \text{dla elementów na drugiej przekątnej} \end{cases}$$

9. Uwagi do projektu nr3.

- Wszystkie tablice alokowane dynamicznie.
- Dane czytamy z pliku.
- Wyniki zapisujemy do pliku i wyświetlamy na monitorze.
- Pełna obsługa błędów dla wszystkich funkcji (patrz plik util_1.cpp z laboratorium).
- Program podzielony na kilka plików (co najmniej 2 pliki).
- W tematach nr 30, 31, 32, 33 i 34 macierz należy wczytać/zapamiętać jako tablice jednowymiarową. W zależności od wybranej opcji funkcja realizuje wskazane zadanie.
W argumentach funkcji możemy stosować parametry domyślne.

10. Przeanalizuj w trybie pracy krokowej przykłady do wykładu 8:

Wszystkie przykłady dostępne pod adresem:

<http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/>

*Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.