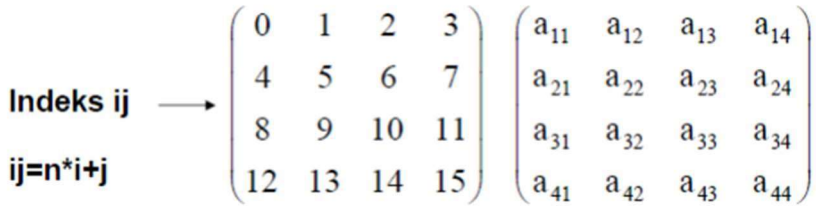
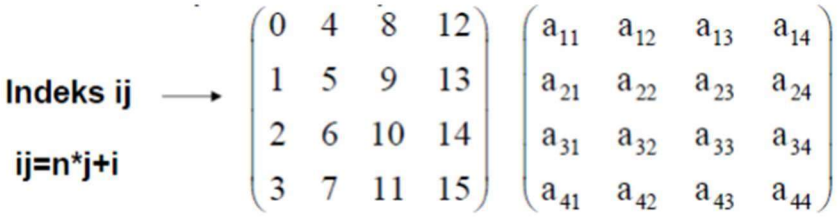
Języki i paradygmaty programowania 1 – studia stacjonarne 2019/20

Lab 8. Tablice liczbowe cd,. Operacje macierzowo-wektorowe, memcpy, memmove, memset. Wyrażenie warunkowe.

1. *Wektory i macierze:*
   a. *Przykład 1. Przechowywanie macierzy (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++)



   b. *Przykład 2. Przechowywanie macierzy (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++)



   c. *Przykład 3. Mnożenie macierzy przez macierz* $c_{ij}=\Sigma k\ a_{ik} \cdot b_{kj}$ *metodą:*
      i klasyczną, czyli “wiersz po wierszu”:

```
for (i = 0; i < M; i++)
for (j = 0; j < N; j++)
for (k = 0; k < K; k++)
```

$C_{ij} = C_{ij} + A_{ik} * B_{kj}$

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**.

      ii Przyspieszoną

```
for (i = 0; i < M; i++)
for (k = 0; k < K; k++)
for (j = 0; j < N; j++)
```

$C_{ij} = C_{ij} + A_{ik} * B_{kj}$

Elementy macierzy A w pętli wewnętrznej nie zależy 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. Funkcje **memcpy, memmove, memset** znajdują się w bibliotece **#include <memory.h>.**

```
void *memcpy(void *dest, const void *src, size_t count);
```
funkcja kopiuje count bajtów z src do dest. Jako wynik zwraca wskaźnik do dest. Jeśli obszar pamięci src i dest pokrywa się działanie funkcji jest nieprzewidywalne.

```
void *memmove(void *dest, const void *src, size_t count);
```
funkcja kopiuje count bajtów z src do dest. Jeśli jakiś obszar pamięci src i dest pokrywa się funkcji gwarantuje, że bajty źródłowe w pokrywającym się regionie zostaną skopiowane zanim zostaną przepisane.

```
void *memset(void *dest, int c, size_t count);
```
Przypisuje symbol c pierwszym count symbolom tablicy dest. Zwraca wskaźnik do dest.

3. Utwórz nowy projekt a następnie skopiuj i przeanalizuj poniższe przykłady użycia funkcji memcpy, memmove, memset.

a) Przykład 1.

```
#include <memory.h>
#include <string.h>
#include <stdio.h>
char str1[7] = "aabbcc";
int main(void)
{
  printf("The string: %s\n", str1);
  //Pokrywający się region! Kopiowanie może nie być poprawne
  memcpy(str1 + 2, str1, 4);
  printf("New string: %s\n", str1);
  strcpy_s(str1, sizeof(str1), "aabbcc"); // reset string
  printf("The string: %s\n", str1);
  //Pokrywający się region! Kopiowanie poprawne
  memmove(str1 + 2, str1, 4);
  printf("New string: %s\n", str1);
}
```

b) Przykład 2.

```
#include <memory.h>
#include <stdio.h>
int main(void)
{
  char buffer[] = "This is a test of the memset function";
  printf("Before: %s\n", buffer);
  memset(buffer, '*', 4);
  printf("After: %s\n", buffer);
}
```

Wynik działania:

Before: This is a test of the memset function

After : **** is a test of the memset function

c) Przykład 3.

```
#include <memory.h>
#include <stdio.h>
int main(void)
 {
  char str[256];
  double aa[200];
  double cc[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  double dd[5] = { 20, 30, 40, 50, 60 };
  memset((void *)str, 0, 256 * sizeof(char)); //OK
```

```
 memset((void *)str, ' ', 256 * sizeof(char)); //!OK – pisanie po pamięci
 memset((void *)str, ' ', 255 * sizeof(char)); //OK
 memset((void *)aa, 0, 200 * sizeof(double)); //OK
 memcpy((void *)&cc[3], (const void *)&dd[2], 2 * sizeof(double));
}
```

Wynik działania:
cc: 1 2 3 40 50 6 7 8 9 10

4. Opracować program, który utworzy dynamicznie (za pomocą funkcji malloc) dwie tablice t1 i t2 liczb całkowitych o rozmiarze n wczytywanym z klawiatury. Jeżeli operacja przydziału pamięci dla t1 i t2 zakończyła się pomyślnie (wskaźniki t1 i t2 są różne od NULL), to zainicjuj tablicę t1 losowymi liczbami. Napisz procedurę, która korzystając z funkcji: **srand**, **time**, **rand** oraz operatora dzielenia modulo uzupełni przekazaną do niej tablicę typu int o losowe liczby z zakresu od 0 do 99. Tablica niech będzie przekazywana przez wskaźnik. Znaleźć minimalny i maksymalny element w tablicy oraz jego położenie. Wyprowadzić na ekran zawartość tablicy t1, a także znalezione wartości i położenie (indeksy) minimum i maksimum. Przekopiować tablicę t1 do t2 wykorzystując standardową funkcję memmove lub memcpy. Wyprowadzić zawartość tablicy t2 na ekran. Przed zakończeniem programu zwolnić pamięć przydzieloną t1 i t2.

5. 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 tych wektorów: $z_i = x_i + y_i$;
   b) Iloczyn skalarny wektorów: $s = \sum_{i=0}^{n-1} x_i \cdot y_i$;
   c) Maksymalną współrzędną wektorów $\vec{x}, \vec{y}$

```
//==============================PLIK_1==================================
#pragma warning (disable:4996)
#include<stdio.h>
#include<stdlib.h>
#define LL 200
extern void error(int, char *);
void argumenty(int, char **);

int main(int argc, char *argv[])
{
	double x[LL], y[LL], z[LL], s, mx, my;
	FILE *fw, *fd;
	int n, k;
	argumenty(argc, argv);
	if (!(fd = fopen(argv[1], "r"))) error(2, "dane");
	if (!(fw = fopen(argv[2], "w"))) error(2, "wyniki");
	fscanf(fd, "%d", &n);
	for (k = 0; k < n; k++)
		fscanf(fd, "%lf", &x[k]);
	for (k = 0; k < n; k++)
		fscanf(fd, "%lf", &y[k]);
	s = 0;
	mx = x[0];
	my = y[0];
	for (k = 0; k < n; k++)
	{
		z[k] = x[k] + y[k];
		mx = x[k] > mx ? x[k] : mx;
		my = y[k] > my ? y[k] : my;
		s += x[k] * y[k];
	}
```

```
        for (k = 0; k < n; k++)
        {
                fprintf(fw, "%lf ", z[k]);
                if (!((k + 1) % 5)) fprintf(fw, "\n");
        }
        fprintf(fw, "\nilocz.skal=%lf mx=%lf my=%lf\n", s, mx, my);
}

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.c ****************************/
#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);
}
```

6. **Wyrażenie warunkowe** przyjmuje postać:
   wyrażenie1 ? wyrażenie2 : wyrażenie3
   wartością wyrażenia warunkowego jest:
   - Wartość wyrażenia2, o ile wyrażenie1 jest różne od zera,
   - Wartość wyrażenia3, o ile wyrażenie1 jest równe zero.

   Operator pytajnik, dwukropek (? :) jest prawostronnie łączny i ma priorytet wyższy jedynie od operatorów przypisania i operatora przecinkowego.

7. Zmodyfikuj program z powyższego przykładu tak aby:
   - Zamiast tablic zdefiniować zmienne typu `double *;`
   - Zarezerwować dokładnie n miejsc na zmienne typu `double` (alokacja pamięci), np.:
     `x = (double*)malloc((unsigned)n * sizeof(double))`

8. Algorytm z pkt.7 podziel na kilka funkcji:
   a) Alokacja pamięci dla tablicy: `double *DajWekt(int n)`
   b) Czytanie elementów tablicy z pliku: `void CzytWekt(FILE *fd, double *we, int n)`
   c) Pisanie elementów tablicy do pliku: `void PiszWekt(FILE *fw, double *we, int n)`
   d) Obliczanie sumy dwóch wektorów:
      `void DodWekt(double *w1, double *w2, double *w3, int n)`

e) Obliczanie iloczynu skalarnego: `double IloczynSkal(double *w1,double *w2,int n)`
f) Obliczanie maksymalnej współrzędnej wektora: `double MaxElem(double *w, int n)`

Funkcje realizujące zadania a), b), c) zapisz w pliku util_2.cpp.
Funkcje realizujące zadania d), e), f) zapisz w pliku util_3.cpp.

9. Przeanalizuj w trybie pracy krokowej przykłady do wykładu 8:
   Wszystkie przykłady dostępne pod adresem:
   http://torus.uck.pk.edu.pl/~fialko/text/CC/przykl/

**Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.*