Temat zajęć: Tablice wielowymiarowe i struktury w języku C

Autor: mgr inż. Sławomir Samolej

Zagadnienie 1. (Tablice liczbowe wielowymiarowe)

Tablicę 2-wymiarową można przedstawić jako pewien zestaw tablic 1-wymiarowych np.:

float tab1[3][4];

tab1 można interpretować jako tablicę 3-elementową złożoną z tablic 4-elementowych.
Tablicę 3-wymiarową można traktować jako zestaw tablic 2-wymiarowych itd.
W przypadku tablicy 2-wymiarowej indeksy można traktować jako numer wiersza i numer kolumny.

Dostęp do elementów tablicy uzyskuje się najprościej przez podanie odpowiedniego zestawu indeksów np.:

```
int a[3][2]=    {       {2,3},
                        {4,8},
                        {1,4}
                };
int c;

a[0][0]=0;      // elementowi o indeksie (0,0) nadaj wartość 0;
c=a[0][1];      // zmiennej c przypisz zawartość elementu tablicy a o indeksach (0,1)
```

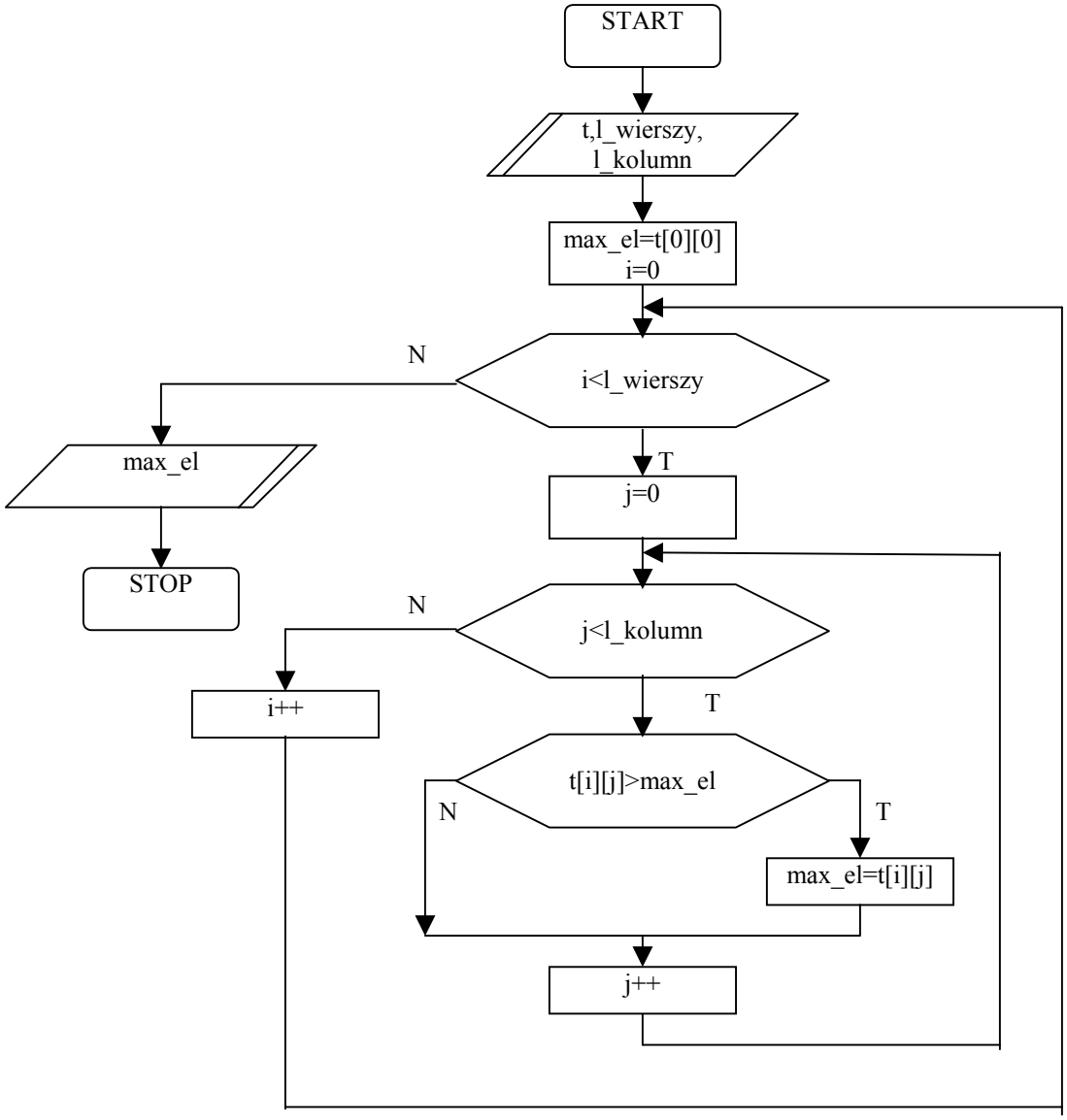
Zadania:

Zadanie 1

Dany jest program:

```
#include <stdio.h>
#define LW 3
#define LK 4

int tab[LW][LK]=    {       {23,45,33,15},
                            {1,-23,32,2},
                            {-22,-24,56,5}
                    };

int max(int t[][LK], int l_wierszy, int l_kolumn);

void main(void)
{
        int i,j;
        int max_el;
        printf("Analizowana tablica:\n");
        for(i=0;i<LW;i++)
        {
                for(j=0;j<LK;j++)
                        printf("%6d",tab[i][j]);
                putchar('\n');
        }
        max_el=max(tab,LW,LK);
        printf("Najwiekszy element analizowanej tablicy: %d\n",max_el);
}

int max(int t[][LK], int l_wierszy, int l_kolumn)
{
        int m_el;

        return m_el;
}
```

Uzupełnić funkcję „max” w taki sposób, aby przeszukała ona tablicę dwuwymiarową „t” o wymiarach „l_wierszy”, „l_kolumn”, znalazła największy element tej tablicy, a następnie zwróciła go. Proponowany algorytm działania funkcji:


START
t,l_wierszy,
l_kolumn
max_el=t[0][0]
i=0
i<l_wierszy
N
T
max_el
STOP
j=0
j<l_kolumn
N
T
i++
t[i][j]>max_el
N
T
max_el=t[i][j]
j++

Zadanie 2

Dany jest program:

```
#include <stdio.h>

int A[3][4]=    {       {1,2,3,4},
                                {5,6,7,8},
                                {9,10,11,12}
                        };
int B[4][2]=    {       {1,2},
                                {3,4},
                                {5,6},
                                {7,8}
                        };
int C[3][2];

void iloczynAB(int A[3][4],int B[4][2], int C[3][2], int m, int n, int p);

void main(void)
{
        int i,j;
        printf("Tablica A:\n");
        for(i=0;i<3;i++)
        {
                for(j=0;j<4;j++)
                        printf("%6d",A[i][j]);
                putchar('\n');
        }
        printf("Tablica B:\n");
        for(i=0;i<4;i++)
        {
                for(j=0;j<2;j++)
                        printf("%6d",B[i][j]);
                putchar('\n');
        }
        iloczynAB(A,B,C,3,4,2);
        printf("Wynik mnozenia A*B:\n");
        for(i=0;i<3;i++)
        {
                for(j=0;j<2;j++)
                        printf("%6d",C[i][j]);
                putchar('\n');
        }
}
```

```
void iloczynAB(int A[3][4],int B[4][2], int C[3][2], int m, int n, int p)
{
}
```

Napisać funkcję „iloczynAB”, która oblicza iloczyn macierzy A(m,n) i B(n,p) i wynik mnożenia zapisuje w macierzy C(m,p). Litery m,n,p oznaczają odpowiednie wymiary macierzy.
Iloczyn macierzy oblicza się zgodnie z zależnością:

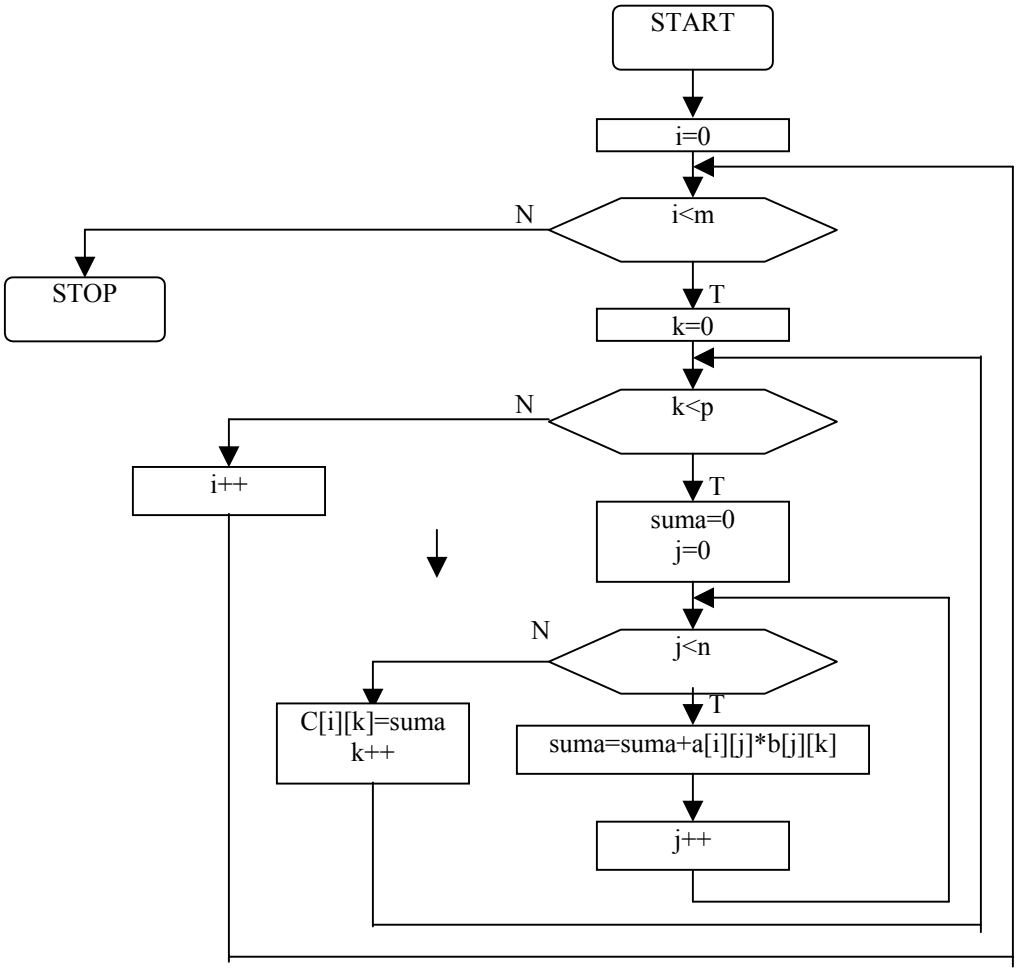
$$C_{i,k} = \sum_{j=0}^{n-1} a_{i,j} \times b_{j,k}$$

$gdzie$:

$i = 0,1,2,...,m-1;$

$j = 0,1,2,...,p-1;$

Schemat blokowy obliczania iloczynu macierzy może wyglądać następująco:



Zagadnienie 2. (Struktury)

Struktura jest obiektem złożonym z jednej lub kilku zmiennych, być może różnych typów.
Struktury można deklarować w następujący sposób:
a) przez zastosowanie słowa kluczowego „struct”:

```
struct point            // deklaracja struktury
{
        int x;
        int y;
};

struct point pt;        // utworzenie obiektu typu struct point
```

b) przez zastosowanie słowa „struct” i metody definiowania nowych typów danych przy pomocy słowa kluczowego „typedef”:

```
typedef struct point            // deklaracja struktury
{
        int x;
        int y;
} PUNKT;

PUNKT pt;               // utworzenie obiektu typu PUNKT
```

Elementom struktury można przypisać początkowe wartości w momencie inicjalizacji:

```
struct point                    // deklaracja struktury
{       int x;
        int y;
};

struct point pt={10,20};        // utworzenie i inicjalizacja obiektu typu struct point
```

Poza inicjalizacją do pól struktury można odwołać się przy pomocy operatora „.”:

```
struct point                    // deklaracja struktury
{       int x;
        int y;
};

struct point pt;                // utworzenie obiektu typu struct point

pt.x=10;                        // nadanie wartości 10 polu x struktury pt.
pt.y=20;
```

Dopuszczalne jest zagnieżdżanie struktur, np.:

```
struct point {int x; int y};
struct rect{struct point pt1; struct point pt2};
```

Aby uzyskać dostęp do odpowiednich pól danych należy posłużyć się kilkakrotnie operatorem ".":

```
struct point {int x; int y};
struct rect{struct point pt1; struct point pt2};

struct rect screen;

screen.pt1.x=5;
```

Można zdefiniować wskaźnik na strukturę i przy jego pomocy odwoływać się do elementów struktury:

```
struct point {int x; int y};      // definicja struktury
struct point *point_ptr;          // definicja wskaźnika
struct point p1;                  // utworzenie obiektu typu "struct point"
int a,b;

point_ptr=&p1;                    // przypisanie wskaźnikowi "point_ptr" adresu struktury „p1"
point_ptr->x=12;                          // zastosowanie operatora "->" do uzyskania dostępu do pola
                                  // struktury
b= p1.y;
a= point_ptr->y;
```

W języku C dopuszczalne jest tworzenie funkcji działających w następujący sposób na strukturach:

a) funkcja może zwracać strukturę w całości np.:

```
struct point makepoint(int x, int y);
```

b) struktura może być argumentem wywołania funkcji:

```
struct point addpoint(struct point p1, struct point p2);
```

c) argumentem wywołania funkcji może być wskaźnik na strukturę:

```
struct point addpoint(struct point *p1_ptr, struct point *p2_ptr);
```

Możliwe jest tworzenie tablic struktur:

```
struct klucz    {       char litera;
                        int licznik;
                };
struct klucz statystyka[]=
{       {‘a’,0},
        {‘b’,0},
        {‘c’,0}
};
```

Zadania:

Zadanie 3

Dany jest program:

```
#include <stdio.h>

struct s_punkt{char c; int x, y;};

void wyswietl1(struct s_punkt punkt);
void wyswietl2(struct s_punkt *ptr);

void main(void)
{
        struct s_punkt p1;
        p1.c='a';
        p1.x=10;
        p1.y=20;
        wyswietl1(p1);
        wyswietl2(&p1);
}
void wyswietl1(struct s_punkt punkt)
{}
void wyswietl2(struct s_punkt *ptr)
{}
```

Uzupełnić zawartość funkcji wyswietl1 i wyswietl2 w taki sposób, aby wypisywały one na ekranie zawartość struktury do nich przekazanej.

Zadanie 4

Uzupełnić wcześniejszy program o funkcję o prototypie:

```
void init(struct s_punkt *ptr, char c, int x, int y);
```

która wypełnia pola struktury typu struct s_punkt wartościami przekazanymi odpowiednio w parametrach c, x, y. Przykładowo:

```
struct s_punkt p2={‘x’,23,45};        // deklaracja i inicjalizacja pewnej struktury

init(&p2, ‘z’, 10, 10);               // wywołanie funkcji init…
                                      // po wykonaniu funkcji struktura p2 powinna być postaci:
                                      // {‘z’,10,10}
```

Zadanie 5

Uzupełnić wcześniejszy program o funkcję o prototypie:

```
struct s_punkt sym(struct s_punkt *ptr);
```

która pobiera dane od struktury typu s_punkt i zwraca strukturę typu s_punkt, ale współrzędne zwracanego punktu są przeciwne do współrzędnych punktu wejściowego. Przykładowo:

```
struct s_punkt p2;                    // deklaracja i inicjalizacja pewnej struktury
struct s_punkt p3;

init(&p2, ‘x’,23,-45);                // wywołanie funkcji init…
p3=sym(&p2);                                  // zawartość struktury p3 powinna wynosić: {‘x’,-
23,45};
```

Zadanie 6

Podczas łamania szyfrów, częstym zabiegiem jest dokonanie analizy częstotliwości występowania poszczególnych liter w tekście. Do zapamiętania ilości występowania liter może posłużyć tablica struktur zaproponowana poniżej (pierwsze pole struktury to kod znaku, drugie – ilość wystąpień w tekście):

```
struct klucz{char litera; int licznik;};

struct klucz statystyka[]=
{
        {'a',0},
        {'b',0},
        {'c',0},
        {'d',0},
        {'e',0},
        {'f',0},
        {'g',0},
        {'h',0},
        {'i',0},
```

```
	{'j',0},
	{'k',0},
	{'l',0},
	{'m',0},
	{'n',0},
	{'o',0},
	{'p',0},
	{'r',0},
	{'s',0},
	{'t',0},
	{'u',0},
	{'w',0},
	{'x',0},
	{'y',0},
	{'z',0},
};
```

Zadanie polega na napisaniu programu, który dokona analizy jednej linii tekstu pod względem częstotliwości występowania poszczególnych liter. Ilość wystąpień poszczególnych liter powinna być zapisana w tablicy struktur „statystyka”. W przyszłości program będzie zastosowany do analizy całych plików tekstowych. Program wypisujący zawartość struktury może mieć postać:

```
#include <stdio.h>
struct klucz{char litera; int licznik;};

struct klucz statystyka[]=
{
	{'a',0},
	{'b',0},
	{'c',0},
	{'d',0},
	{'e',0},
	{'f',0},
	{'g',0},
	{'h',0},
	{'i',0},
	{'j',0},
	{'k',0},
	{'l',0},
	{'m',0},
	{'n',0},
	{'o',0},
	{'p',0},
	{'r',0},
	{'s',0},
	{'t',0},
```

```
        {'u',0},
        {'w',0},
        {'x',0},
        {'y',0},
        {'z',0},
};

void main(void)
{
        int i;
        for(i=0;i<24;i++)
                printf("%4c,%5d\n",statystyka[i].litera, statystyka[i].licznik);
}
```