2D static arrays

Declaring a 2D static array works similar as working with 1D static arrays.

Declaration needs:

- type
- name - identifier like any other variable
- size_1 and size_2 - **MUST** be known at compilation time

- Storage is continous in memory - see slides

- Occupies size 1 x size 2 x sizeof(type) B
- element acces with double square brackets: [][]

int tabA[2][3]; - declares an array of 6 int double tabB[5][3]; - an array of 15 doubles

Some examples:

Declare a 2D array of ints and fill it with data using [][], than print some values

In [1]:
```c
#include <stdio.h>

int main()
{
    int a[4][3]; // 4 x 3 array of ints

    a[0][0] = 0;
    a[0][1] = 1;
    a[0][2] = 2;

    a[1][0] = 10;
    a[1][1] = 11;
    a[1][2] = 12;

    a[2][0] = 20;
    a[2][1] = 21;
    a[2][2] = 22;

    a[3][0] = 30;
    a[3][1] = 31;
    a[3][2] = 32;

    printf("%d %d %d", a[0][0], a[1][1], a[3][2]);
}
```

```
0 11 32
```

We will now modify the program a bit, to illustrate that the storage in memory is continuous. W do this by retrieving the address of the first element and than by increasing it by one.

In [5]:
```c
#include <stdio.h>

int main()
{
    int a[4][3]; // 4 x 3 array of ints

    a[0][0] = 0;
    a[0][1] = 1;
```

```
    a[0][2] = 2;

    a[1][0] = 10;
    a[1][1] = 11;
    a[1][2] = 12;

    a[2][0] = 20;
    a[2][1] = 21;
    a[2][2] = 22;

    a[3][0] = 30;
    a[3][1] = 31;
    a[3][2] = 32;

    int *p = &a[0][0];

    printf("%d\n", *p);
    ++p;
    printf("%d\n", *p);
    ++p;
    printf("%d\n", *p);
    ++p;
    printf("%d\n", *p);
    ++p;
    printf("%d\n", *p);
    ++p;
    printf("%d\n", *p);
}
```

```
0
1
2
10
11
12
```

Print addresses of first elements in each row, what is the distance between them?

In [6]:

```
#include <stdio.h>

int main()
{
    int a[4][3];

    printf("%p %p\n", &a[0][0], &a[0][1]);
    printf("%p %p\n", &a[1][0], &a[1][1]);
}
```

```
0x7ffcde4e3fc0 0x7ffcde4e3fc4
0x7ffcde4e3fcc 0x7ffcde4e3fd0
```

Conclusion: storage is continuous and elements are separated by row size

In [7]:

```
#include <stdio.h>
#define X 500
#define Y 100

int main()
{
    int a[X][Y]; // larger array

    int x = 5; // actual size of a 2D array
    int y = 10;

    // .. work with this array
```

```
    printf("%p %p\n", &a[0][0], &a[0][1]);
    printf("%p %p\n", &a[1][0], &a[1][1]);
}
```

```
0x7ffe70fa5bf0 0x7ffe70fa5bf4
0x7ffe70fa5d80 0x7ffe70fa5d84
```

We see that the row size (the number of columns) is important. The reason for that is that by changing the value in the second bracket we move by a single pointer (single address), by changing the value in the first bracket we need to move by the entire row. This value can only be known if we know exactly how many columns there are!

A consequence is that a 2D array be treated as a 1D array in some situations but not in others.

1. Storage is continuous!

In [8]:

```
#include <stdio.h>

int main()
{
    int a[4][3];
    int *p = &a[0][0]; // adraess and an array are the same?

    for(int i=0; i<10; ++i)
        p[i] = i;

    printf("%d %d %d", a[2][0], a[2][1], a[2][2]);
}
```

```
6 7 8
```

Can the 1D array be treated as a 2D? Yes, and no. There is a but. So the size of a row would have to be known.

When writing functions working with 2D arrays we need to remember to provide the second dimension!

Write a function that fills and prints the content of a 2D array of integers:

In [19]:

```
#include <stdio.h>

void fill(int tab[10][10], int n, int m);
void print(int tab[10][10], int n, int m);

int main()
{
    int tab[10][10];
    int n=6, m=3; // rows and collumns
    fill(tab, n, m);
    print(tab, n, m);
}

void fill(int tab[10][10], int n, int m)
{
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
        {
            tab[i][j] = 100*i+10*j;
        }
    }
```

```
}

void print(int tab[10][10], int n, int m)
{
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
        {
            printf("%d \t", tab[i][j]);
        }
        printf("\n");
    }
}
```

```
0       10      20
100     110     120
200     210     220
300     310     320
400     410     420
500     510     520
```

A different way to initialize arrays:

1D

In [20]:

```
#include <stdio.h>
int main(){
    int a[] = {1,2,3};
    printf("%d %d %d", a[0], a[1], a[2]);
}
```

```
1 2 3
```

2D is also possible, but as with functions the size of a row (number of collumns must be known).

In [21]:

```
#include <stdio.h>
int main(){
    int a[][2] = {1,2,3,4};
    printf("%d %d %d %d", a[0][0], a[0][1], a[1][0], a[1][1]);
}
```

An example

- Write a program illustrating workings of a 2D static array
- Add initialization function
- Distinguish the maximum size of an array, and the one used by the program
- Illustrate how to write functions with 2D arrays
- Add a function printing a 2D array
- Add a function coping to a 1D vector the diagonal from a square matrix
- Write a function coping a row, column from a 2D array
- Write a function inserting a row column into a 2D array

In [16]:

```
#include <stdio.h>
#define MAX_SIZE 100

void fill(int tab[MAX_SIZE][MAX_SIZE], int n, int m);
void print(int tab[MAX_SIZE][MAX_SIZE], int n, int m);
void diag(int tab[][MAX_SIZE], int n, int d[]);
void print1D(int tab[], int n);
```

```c
int* copy1(int tab[][MAX_SIZE], int n);
void copy(int tab[][MAX_SIZE], int n, int row, int d[]);
void insert_collumn(int tab[][MAX_SIZE], int n, int col, int d[]);

int main()
{
    int tab[MAX_SIZE][MAX_SIZE];
    int n=6, m=6; // rows and collumns

    fill(tab, n, m);
    print(tab, n, m);
    printf("--------\n");

    int d[MAX_SIZE];
    diag(tab, n, d);
    print1D(d, n);
    printf("--------\n");

    int *p = copy1(tab, 3);
    print1D(p, n);

    copy(tab, n, 2, d);
    print1D(d, n);
    printf("--------\n");
    p[1] = 1000;
    d[1] = 1000;
    print(tab, n, m);
    printf("--------\n");
    insert_collumn(tab, n, 1, d);
    insert_collumn(tab, n, 3, p);
    print(tab, n, m);
}

void fill(int tab[][MAX_SIZE], int n, int m)
{
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
        {
            tab[i][j] = 100*i+10*j;
        }
    }
}

void print(int tab[][MAX_SIZE], int n, int m)
{
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
        {
            printf("%d \t", tab[i][j]);
        }
        printf("\n");
    }
}
void print1D(int tab[], int n)
{
    for(int i=0; i<n; ++i)
    {
        printf("%d \t", tab[i]);
    }
    printf("\n");
}

void diag(int tab[][MAX_SIZE], int n, int d[])
{
```

```
    for(int i=0; i<n; ++i)
    {
        d[i] = tab[i][i];
    }
}

//An address of the first element of a row
int* copy1(int tab[][MAX_SIZE], int n)
{
    return &tab[n][0];
}

void copy(int tab[][MAX_SIZE], int n, int row, int d[])
{
    for(int i=0; i<n; ++i)
    {
        d[i] = tab[row][i];
    }
}

void insert_collumn(int tab[][MAX_SIZE], int n, int col, int d[])
{
    for(int i=0; i<n; ++i)
    {
        tab[i][col] = d[i];
    }
}
```

```
0       10      20      30      40      50
100     110     120     130     140     150
200     210     220     230     240     250
300     310     320     330     340     350
400     410     420     430     440     450
500     510     520     530     540     550
--------
0       110     220     330     440     550
--------
300     310     320     330     340     350
200     210     220     230     240     250
--------
0       10      20      30      40      50
100     110     120     130     140     150
200     210     220     230     240     250
300     1000    320     330     340     350
400     410     420     430     440     450
500     510     520     530     540     550
--------
0       200     20      300     40      50
100     1000    120     1000    140     150
200     220     220     320     240     250
300     230     320     330     340     350
400     240     420     340     440     450
500     250     520     350     540     550
```

In [ ]: