Computer Science I

Lecture 9
2D Arrays

and examples

2D arrays

Declaration

Syntax:

```
type name[size 1][size 2]
```

- *type* - almost any type, pointer, etc.
- *name* - an identifier
- *size1* and *size 2* - **MUST** be known at compilation time

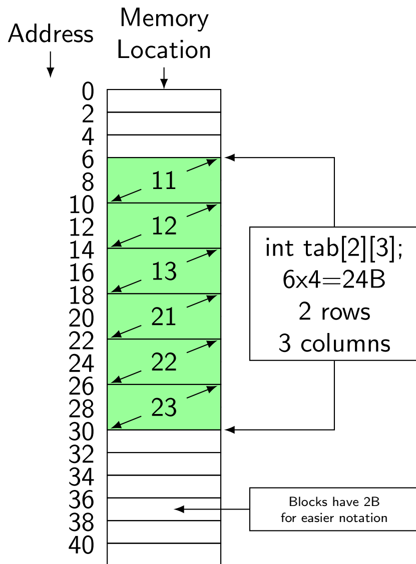
e.g.:

```
//2x3 ints array of-> 6 ints -> 24B
int tabA[2][3];
//5x3 double array ->  15 doubles -> 120B
double tabB[5][3];
```

- Continous in memory
- Occupies *size 1* x *size 2* x *sizeof(type)* B
- Acces elements with double *[]*, e.g.: *tab[i][j]*

```
tabA[0][0] // first row first column
tabA[0][2] // first row third column
tabA[1][2] // second row third column
```

2D arrays
Memory



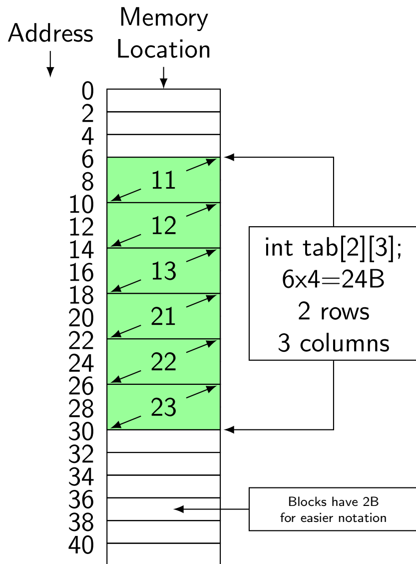
- Indexing is from 0 to size-1
- Storage is row based
- Array is stored row after row

Example:

```
int tab[2][3];

tab[0][0]=11;
tab[0][1]=12;
tab[0][2]=13;

tab[1][0]=21;
tab[1][1]=22;
tab[1][2]=23;
```

11	12	13
21	22	23

2D arrays
Memory



1. Write a program using a 2D static array.
2. Access elements using *[]*.
3. Print an address of each array element using &tab[i][j].
4. What is a distance of:
   - tab[i][j] and tab[i+1][j]
   - tab[i][j] and tab[i][j+1]
5. Can a 2D array be treated as 1D?
6. Consequences?

2D arrays
Functions

- Defining a function:

```
type function_name(array_type tab[][SIZE2], ...)
{
  //Function body
}
```

- Usage:

```
array_type tab[SIZE1][SIZE2];
...
//Call the function, pass an array as an argument
function_name(tab, ...);
```

- The second bracket **MUST** give the size of an array.
- Function is compiled separately
- Changing the second index moves to the next memory block
- Changing the first index moves us to the next row.
- The size of row must be known!
- See previous example!

2D arrays

Examples

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

Dynamic memory allocation

We know how to declare static arrays. We need a method to deal with situations when the size of an array is unknown at compilation time. C offers *malloc*, located in stdlib.h

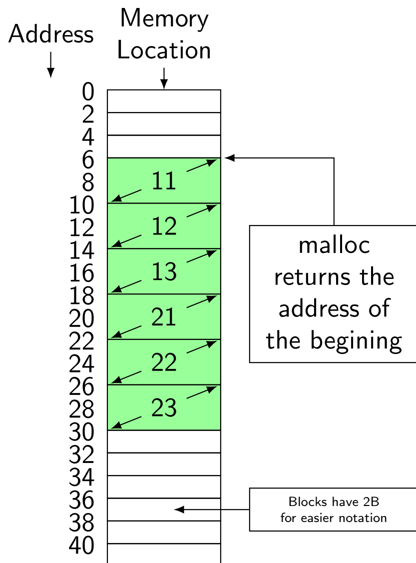
```
void* malloc (size_t size);
void* malloc (unsigned int size);
```

1. Allocates a block of **size** bytes of memory
2. Returns a pointer to the beginning of that block
3. The content allocated block of memory is not initialized
4. *size t* is *unsigned int*
5. For each malloc there needs to be a single *free*

```
type * p = (type*)malloc(size);
free(p)
```

6. **After** we are done with using the memory

Dynamic memory allocation



- Indexing is from 0 to size-1
- Just like the 1D static one

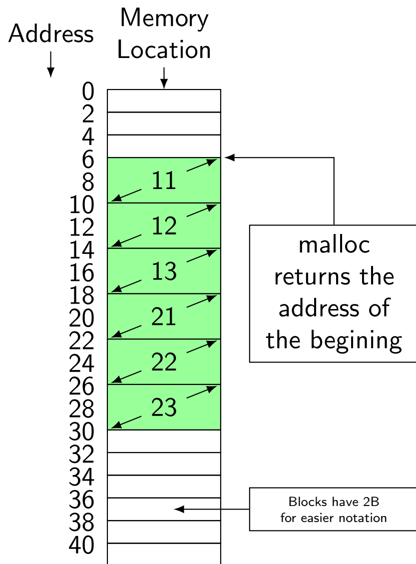
Example:

```
int  *p=(int*)malloc(24);

p[0]  =  11;
p[1]  =  12;
p[2]  =  13;
p[3]  =  21;
p[4]  =  22;
p[5]  =  23;

free(p);
```

Computer Science I

Dynamic memory allocation

use sizeof()



- *sizeof()* gives us the size of type
-

Example:

```
int *p=(int*)malloc(6*sizeof(int));

p[0] = 11;
p[1] = 12;
p[2] = 13;
p[3] = 21;
p[4] = 22;
p[5] = 23;

free(p);
```

Dynamic memory allocation

With size from keyboard

- Read the size from keyboard
- Allocate memory using *malloc()*

```
int n;
scanf("%d", &n);
int *p=(int*)malloc(n*sizeof(int));

p[0] = 11;
p[1] = 12;
p[2] = 13;
p[3] = 21;
p[4] = 22;
p[5] = 23;

free(p);
```

- Recall the example with reading in array data from a file
- Read the size from file, allocate, read data ...

Computer Science I

Dynamic memory allocation

Use with functions and compatibility with static arrays

- In the case of 1D arrays it is the same as with static ones
- Example with bubble sorting

Dynamic memory allocation

Allocation of 2D arrays is a bit more complicated ...

Which we will find out next week