CMSC 21
Fundamentals of Programming

2nd Semester 2011-2012

MULTIDIMENSIONAL ARRAYS

Multidimensional Arrays

- Arrays with more than one dimension

```
<data_type> <var_name>[size1][size2]…[sizen];

//two-dimensional array of integers
int table [3][3];

//three-dimensional array of float
float rgb [3][4][5];
```

Initializing Multidimensional Arrays

- To initialize two-dimensional arrays,

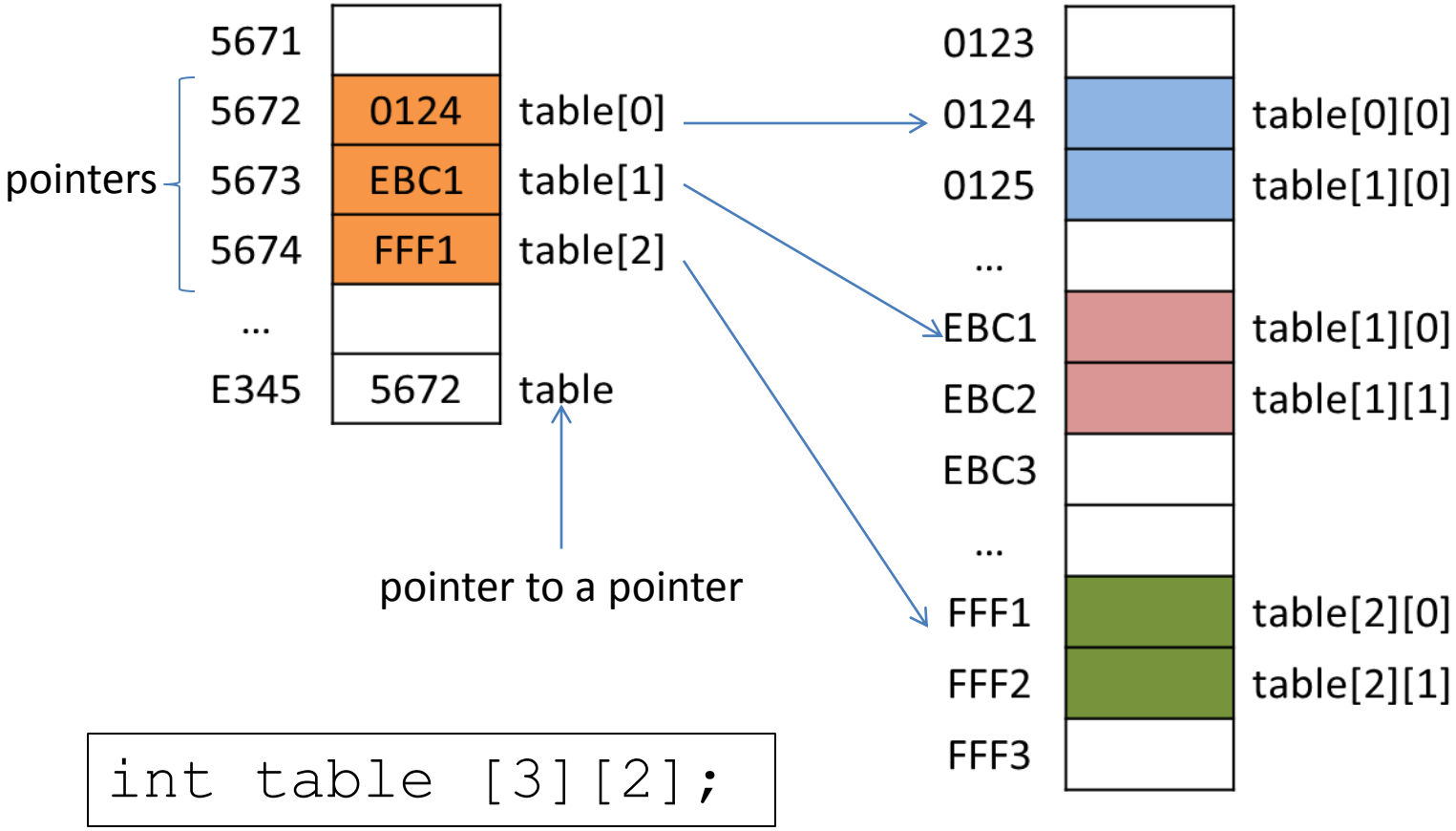
```
int table[3][4]={{1,2,3,4},
 {5,6,7,8}, {9,10,11,12}};
```

1	2	3	4
5	6	7	8
9	10	11	12

2D Arrays and the Memory

- The variable name of a two-dimensional array is a pointer to an array of pointers
- Each pointer holds the address of the first element of an array of type `<data_type>`

2D Arrays and the Memory




```
int table [3][2];
```

Accessing Multidimensional Arrays

- Indexing
  - Just like in one-dimensional array
  - Specify the index inside []
  - The index is also from 0 … size-1
  - Example:

```
//assign 51 to the 1st row, 2nd col
table[0][1] = 51;
```

Accessing Multidimensional Arrays

- Pointer Arithmetic
  - Use pointers to pointers to access array elements
  - Pointer arithmetic can be performed on the array name
  - Example:

```
//assign 51 to the 1st row, 2nd col
*(*(table+0)+1) = 51;
```

Accessing Multidimensional Arrays

- In general, to access multidimensional arrays:
  - Indexing

    <var_name>[i_1][i_2][i_3]…[i_n]

  - Pointer Arithmetic

    (…(*(<var_name>+i_1)+i_2)+…+i_n)

Multidimensional Arrays as Parameters

- To pass a multidimensional array to a function, the name of the array is specified as the actual parameter
- As formal parameter:
  - Specify all the sizes
  - Specify the sizes except for the first dimension

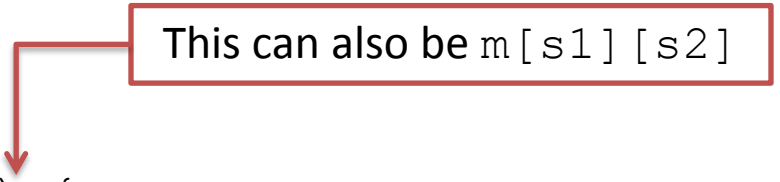
Multidimensional Arrays as Parameters

- Example:

```
#include <stdio.h>
#define s1 3
#define s2 4

void initialize (int m[][s2]) {
     int i, j;
     for (i = 0; i<s1; i++)
         for (j=0; j<s2; j++) m[i][j] = i+j;
}

int main () {
    int m[s1][s2];
    initialize(m);  //pass the name of the array
}
```



Multidimensional Arrays as Parameters

- Dimensions other than the first one should be specified so that the compiler can determine, within the function, the depth of each additional dimension

DYNAMIC ARRAYS

Dynamic Arrays

- Dynamic Arrays are arrays that are allocated in the memory at runtime
- The size of the array can be set during the execution of the program

Dynamic Arrays

- Dynamic Arrays are allocated in the memory using:
  - Pointers
  - `malloc` function

The `malloc` function (1)

- malloc function is used to allocate a specific amount of memory during the execution of a program.

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

- `size_t` is an unsigned integer returned by the operator sizeof

The `malloc` function (2)

- If the memory allocation is successful, a pointer to the memory block allocated by the function is returned
- Otherwise, a **null pointer** is returned
- Include `stdlib.h` to use `malloc` function

`void *`

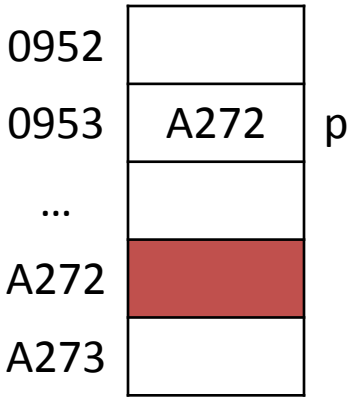
- The type of pointer returned by the `malloc` function is always `void *`
- This pointer can be typecasted to the desired type of data

The `malloc` function (3)

- Example:

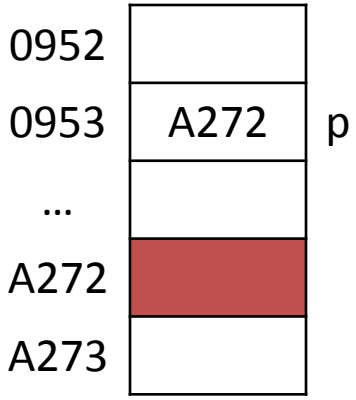
```
#include <stdlib.h>
main () {
    int *p;
    //allocate one dynamic integer in the memory
    p = (int *) malloc (sizeof(int));
}
```

Typecast `void*` to a pointer to an integer since we wish to allocate integer in the memory



The `malloc` function (4)

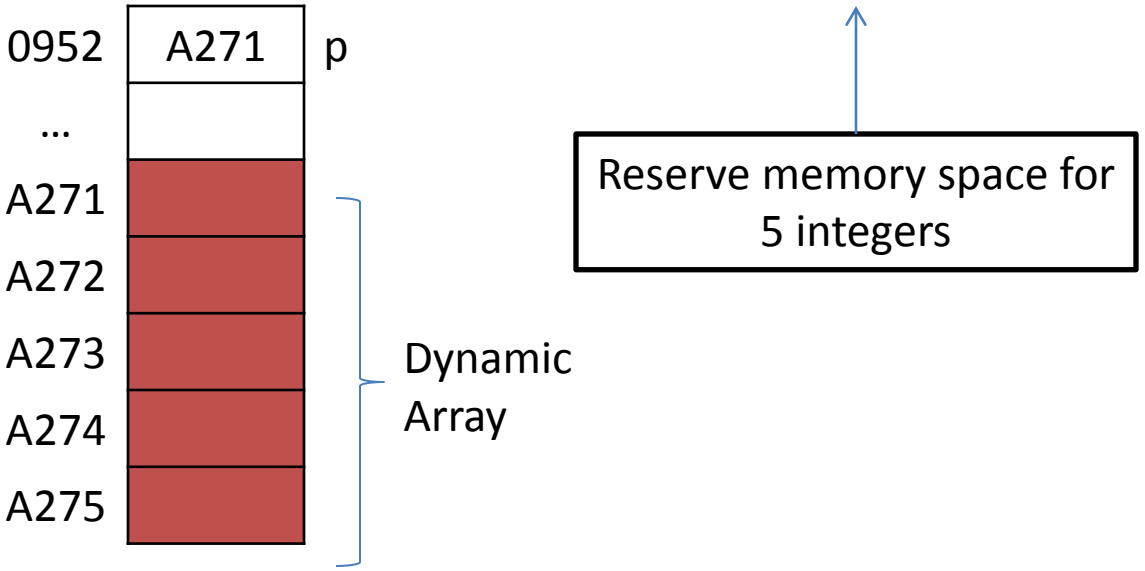
- The newly created dynamic variable has no name or identifier
- Dynamic variables can be accessed using pointers that hold their addresses



Dynamic one-dimensional Arrays

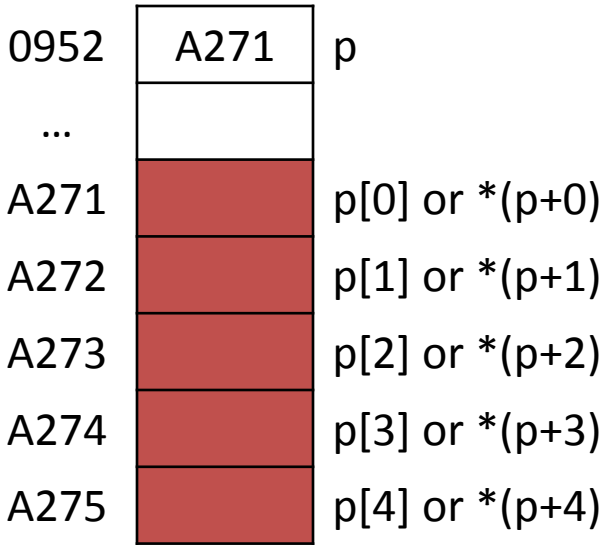
- Example:

```
#include <stdlib.h>
main () {
    int *p, size = 5;
    //allocate 5 dynamic integers in the memory
    p = (int *) malloc (size * sizeof(int));
}
```



Dynamic one-dimensional Arrays

- The allocated space is an array of 5 integers
- Allocated integers can be accessed using either indexing or pointer arithmetic



Dynamic Multidimensional Arrays

- Dynamic multidimensional arrays are allocated using pointers to pointers
- A dynamic two-dimensional array uses a pointer to a pointer
- A dynamic three-dimensional array uses a pointer to a pointer to a pointer
- An so on…

Dynamic Multidimensional Arrays

- What happens when a multidimensional array is allocated dynamically?
  - When a two dimensional array is allocated:
    - Declare a pointer to a pointer
    - Allocate pointers equivalent to the number of rows
    - For each pointer, allocate memory spaces for the values equivalent to the number of columns

Dynamic Multidimensional Arrays

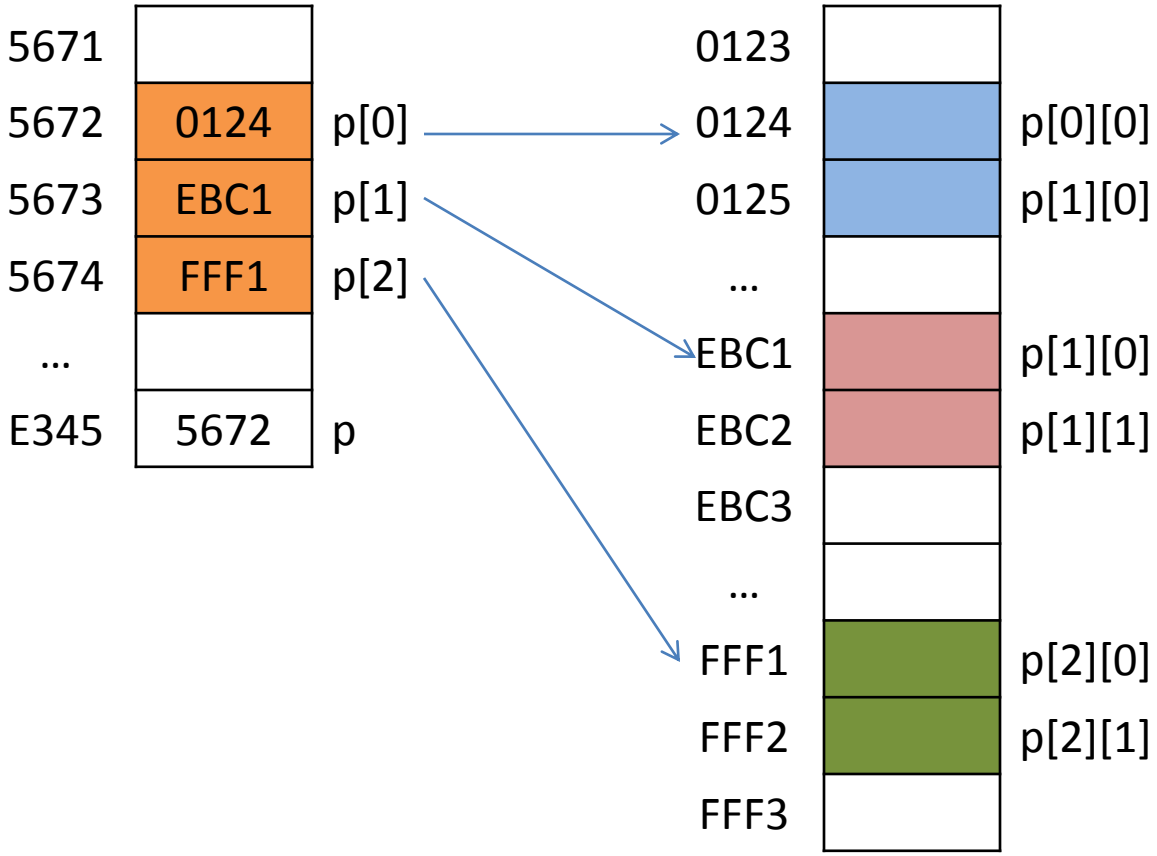
- What happens when a multidimensional array is allocated dynamically?
  - When a three dimensional array is allocated:
    - Declare a pointer to a pointer to a pointer
    - Allocate pointer to a pointer equivalent to the first size
    - For each pointer to a pointer, allocate memory spaces for pointers equivalent to the second size
    - For each pointer, allocate memory spaces for the values, equivalent to the third size

Dynamic Two-Dimensional Arrays

- Example:

```
main () {
    int row = 3, col = 2, **p, i;
    //alocate space for pointers
    p = (int **) malloc (row*sizeof(int*));
    for (i = 0; i<row; i++)
        //allocate space for array elements
        *(p+i)=(int *)malloc(col*sizeof(int));
        //*(p+i) can be p[i]
}
```

Dynamic Two-Dimensional Arrays



Dynamic Arrays as Parameters

- To pass a dynamic array to a function, the name of the pointer is specified as actual parameter
- If the dynamic array is 1D, then the formal parameter is a pointer, if 2D, then pointer to a pointer, and so on…

Quiz (1/4)

1. How many memory spaces does a one-dimensional dynamic array of size 3 occupy?
2. How about a 2-dimensional array with 5 row and 4 columns?
3. How about a 4x4x4 array?