

ACS126 Computing and Systems Design

Lecture 13. Arrays and functions, multi-dimensional arrays



Module roadmap

← C labs →

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

Arrays and Strings,
Pointers,
Modular program
design, Functions
and passing
parameters

Arrays and
pointers,
Multi-
dimensional
arrays,
Dynamic
memory,
allocation
Structures,
Arrays of
structures

Switch-
case,
Recursion,
Putting it
all together
– using
tools for
problem
solving.

Module
review
and
exam
pre-
paration

Robot software due

C program due

C program due

Exam

Learning Outcomes

By the end of this session, students should be able to:

- Pass arrays as parameters to functions
- Manipulate 2D arrays

An Array in Memory

- When an array is declared, a sequential, consecutive block of memory is reserved for it.
- The space needed for each array element will be determined by the type: typically char needs 1 byte, int and float need 4 bytes, double needs 8 bytes. 1 byte is 8 bits. Memory addresses are at the byte level, but we can conceptualise array storage as a set of elements:

```
int list[6] = {1, 3, 5, 7, 9, 0}
```

list[0]	1
list[1]	3
list[2]	5
list[3]	7
list[4]	9
list[5]	0

Pointer and Arrays

- The standard way to access individual elements of an array is with an index: `list[3];` **the index**

- You can also use pointers:

```
int list[6]={1,3,5,7,9,0};
```

```
int *listptr;
```

```
listptr=list;
```

`listptr+0`

`listptr+1`

`listptr+2`

`listptr+3`

`listptr+4`

`listptr+5`

1

3

5

7

9

0

- You are allowed to set `listptr` equal to `list` because both are of type “pointer to int”. **An array name is a pointer to the first element in the array.**



The
University
Of
Sheffield.

PASSING ARRAYS AND STRINGS AS PARAMETERS TO FUNCTIONS

Arrays as Parameters

- When an array is a parameter to a function, remember that the name of the array, with no indices specified, is the address of the first element in the array.

Passing Arrays as Parameters to Functions

Three methods:

1) **Pass each array element**, by value.

E.g.: `void inputArrayElement(int element);`

2) **Pass the array name**. The array name is a pointer to the first element in the array. The function prototype must include `[]` after the array name.

E.g.: `float inputA(int A[], int sizeA);`

3) **Pass the array address** using a pointer.

E.g.: `void inputArrayAddress(int *p1);`

Note: For a string, the array type is `char`!

Pass by Value

```
float PrintMarks(float m, int i); /* Pass the
    array element by element, return array element. */
```

```
main()
```

```
{
```

```
    float marks[]={5,4,3,4,4}, size=0;
```

```
    int i;
```

```
    for(i=0;i<5;i++){
```

```
        size = PrintMarks(marks[i],i);}
```

```
}
```

```
float PrintMarks(float m, int i)
```

```
{
```

```
    printf("Element %d is %0.2f\n",i,m);
```

```
    return m;
```

```
}
```

Declaring the array

Calling the
function

Pass the Array Name

```
float AddMarks(float m[]); /* Pass the array name- i.e. a pointer;  
return total of the list. */
```

```
main()  
{  
    float marks[]={5,4,3,4,4}, size=0;  
    size = AddMarks(marks);  
}  
float AddMarks(float m[])  
{  
    float total=0;  
    int i;  
    for(i=0;i<5;i++)  
    {  
        total = total + m[i];  
    }  
    return total;  
}
```

Declaring the array

Calling the function

Pass by Address/Pointer

```
float InputMarks(float *m); /* Pass the array name- i.e. a  
pointer; return total of the list. */
```

```
main()
```

```
{
```

```
    float marks[]={5,4,3,4,4}, size=0;
```

```
    size = InputMarks(marks);
```

```
}
```

```
float InputMarks(float *m)
```

```
{
```

```
    float total=0;
```

```
    int i;
```

```
    for(i=0;i<5;i++)
```

```
    {
```

```
        total = total + *(m+i);
```

```
    }
```

```
    return total;
```

```
}
```

Declaring the array

Calling the function

Example with a String

```
void printString(char a[]);
```

```
int main() {
```

```
    char s[20];
```

```
    printf("Enter string <19\n");
```

```
    scanf("%s", s);
```

```
    printf("Printing from main: %s\n", s);
```

```
    printString(s);
```

```
}
```

```
void printString(char a[]) {
```

```
    printf("Printing from function: %s\n ", a);
```

```
}
```



The
University
Of
Sheffield.


	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

TWO-DIMENSIONAL ARRAYS


Two Dimensional Arrays

- A **one dimensional array** of numbers is a **vector**.
- A **two dimensional array** of numbers is a **matrix**, having both rows and columns:

`int matrix[2][3];`
ALWAYS!



number of rows



number of columns

- Two dimensional arrays are stored in row order, as shown below:

`int matrix[2][3]={2, 4, 6, 3, 5, 8};`

$$\begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 8 \end{bmatrix}$$

Initialise in
row order.

`matrix[0][0]`
`matrix[0][1]`
`matrix[0][2]`
`matrix[1][0]`
`matrix[1][1]`
`matrix[1][2]`

2
4
6
3
5
8

Working with Two Dimensional Arrays

- It is very common to access two dimensional arrays in a pair of nested for loops, with the **outer loop index controlling the row**, and the **inner loop index controlling the column**, e.g.:

```
for(i=0; i<2; i++)  
{  
    for(j=0; j<3; j++)  
    {  
        printf("%d ", matrix[i][j]);  
    }  
    printf("\n");  
}
```

#define

- Creates a named constant with some value, which the compiler will use to replace every occurrence of that name with the value.
- Allows for easy change to values that occur several places in the program

Example

```
#define numColumns 3
```

```
#define numRows 2
```

```
void main() {
```

```
    int myArray[numRows][numColumns];
```

```
}
```



#define is used
outside main()

17 #include <stdio.h>

#define ROWS 5

#define COLS 5

main()

{

int i, j;

int matrix[ROWS][COLS];

printf("enter matrix elements for a %d by %d matrix
\n", ROWS, COLS);

for(i=0; i<ROWS; i++){

for(j=0; j<COLS; j++){

scanf("%d", &matrix[i][j]);}

}

printf("you entered this matrix:\n");

for(i=0; i<ROWS; i++){

{

for(j=0; j<COLS; j++){

{

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

Example


Test: what would be printed by this code?

```
#include <stdio.h>
#define ROWS 3
#define COLS 2
main()
{
    int i, j;
    float matrix[ROWS][COLS]={1.0, 2.0, 3.0, 4.0,
5.0, 6.0};
    for(i=0; i<COLS; i++)
    {
        for(j=0; j<ROWS; j++)
        {
            printf("%0.1f ",matrix[j][i]);
        }
        printf("\n");
    }
} //end main
```

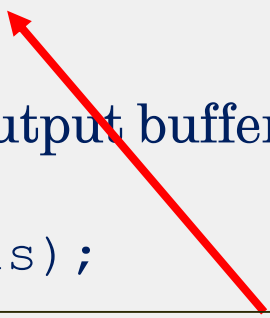
Passing a 2D Array as a Parameter to a Function

- The function prototype and the function header must state the second dimension (i.e. the number of columns).
- The reason for this is that a 2D array is regarded as an array of arrays (i.e. each row is an array). The compiler wouldn't know how to index into the second level of the array unless it knew the extent of memory needed for each. The number of columns gives the size of the inner arrays.
- Example on next slide.

```
#include <stdio.h>
#define ROWS 3
#define COLS 2
void printArray(float arr[][COLS],int rows, int cols);
main()
{
    float matrix[ROWS][COLS]={1.0, 2.0, 3.0, 4.0, 5.0,
6.0};
    printArray(matrix,ROWS,COLS);
} //end main
void printArray(float arr[][COLS],int rows, int cols)
{
    int i,j;
    for(i=0; i<rows; i++)
    {
        for(j=0; j<cols; j++)
        {
            printf("%.1f ",arr[i][j]);
        }
        printf("\n");
    }
}
```

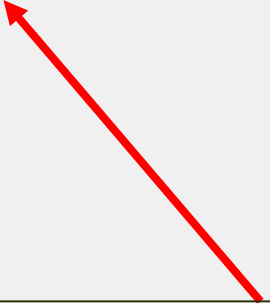


```
#define Cols 5
#define Rows 2
void outputStrings(char a[][Cols],int r,int c);
main()
{
    char myString[Rows][Cols]={};
    int i,j;
    printf("Rows is %d, Columns is %d\n", Rows, Cols);
    for(i=0; i<Rows; i++)
    {
        printf("for row %d, enter %d characters\n", i, Cols);
        for(j=0; j<Cols; j++)
        {
            myString[i][j]=getchar();
        }
        fflush(stdin); // flushes the output buffer of a stream.
    }
    outputStrings(myString,Rows,Cols);
}
```



Or use: `scanf ("%c", &myString[i][j]);`

```
void outputStrings(char a[][Columns],int r,int c)
{
    int i,j;
    printf("outputting string:\n");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        {
            putchar(a[i][j]);
        }
        printf("\n");
    }
}
```



Or use: `printf("%c", a[i][j]);`

Learning Outcomes

By the end of this session, students should be able to:

- Pass arrays as parameters to functions – either as a pointer to the start of the array, or element by element,
- Manipulate 2D arrays –
 - rows x columns,
 - stored in row order- specify number of columns when used in functions
 - initialising,
 - using nested loops to manipulate.