CL 101 INTRODUCTION TO COMPUTING

Fall 2017

LAB 09 MANUAL

RECURSION AND ARRAYS

FAST- NUCES Karachi, Pakistan.

Recursion and Arrays

Recursion:

When a function invokes itself, the call is known as a recursive call. Recursion (the ability of a function to call itself) is an alternative control structure to repetition (looping). Rather than use a looping statement to execute a program segment, the program uses a selection statement to determine whether to repeat the code by calling the function again or to stop the process. Each recursive solution has at least two cases: **the base case** and **the general case**. The base case is the one to which we have an answer; the general case expresses the solution in terms of a call to itself with a smaller version of the problem. Because the general case solves a smaller and smaller version of the original problem, eventually the program reaches the base case, where an answer is known and the recursion stops.

For example, a classic recursive problem is the factorial. The factorial of a number is defined as the number times the product of all the numbers between itself and 0: N! = N * (N-1)! The factorial of 0 is 1. We have a base case, Factorial (0) is 1, and we have a general case, Factorial (N) is N * Factorial (N-1). An if statement can evaluate N to see if it is 0 (the base case) or greater than 0 (the general case). Because N is clearly getting smaller with each call, the base case is reached.

```
#include <stdio.h>
int factorial(unsigned int n) {
   if(n = = 0) {
      return 1;
   }
   return n * factorial(n - 1);
}
void  main() {
   int n = 12;
   printf("Factorial of %d is %d\n", n, factorial(n));
 }
```

What happens if the argument is a negative number? The function just keeps calling itself until the run-time support system runs out of memory. This situation is called infinite recursion and is equivalent to an infinite loop.

Recursion is a very powerful and elegant tool. However, not all problems can easily be solved recursively, and not all problems that have an obvious recursive solution should be solved recursively. But there are many problems for which a recursive solution is preferable. If the problem statement logically falls into two cases, a base case and a general case, recursion is a viable alternative.

Arrays:

An array is a compound data type or a named collection of homogeneous items in which individual items are accessed by their place within the collection. The place within the collection is called an index. All the item in the array must be of the same type, but you can create arrays of integers, arrays of characters and the like.

Array Declaration:

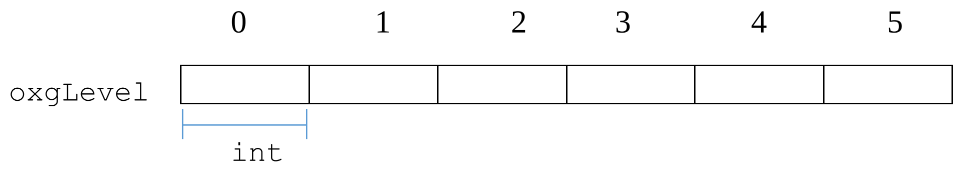
An array declaration specifies the name of the array, the type of elements stored in the array, and the number of array elements (the size of the array).

type name [elements];

For example, six values of type int can be declared as an array without having to declare 6 different variables (each with its own identifier). Instead, using an array, the six int values are stored in contiguous memory locations, and six can be accessed using the same identifier, with the proper index.

int oxgLevel [6];

The declaration creates an array of integers named oxgLevel and reserves enough memory to store six integers.



Where each blank panel represents an element of the array. In C, the first element in an array is always numbered with a zero (not a one), no matter its length. You refer to an individual array element by placing its number (called its subscript or array index) within brackets immediately after the array name.in the oxgLevel array you denote the first element oxgLevel [0], the second element oxgLevel [1] , and so on.

```
//reads six integers into array oxgLevel and displays them to
//the screen three per line.

#include <stdio.h>

void main(void)
{
     int oxgLevel[6]; // array of six oxygen level readings
     int j;              // loop control index

     // input six integers into the oxygen level array.
     printf(" Enter the six oxygen level readings: ");
     for (j=0 ;j<6 ;j++)
     scanf("%d",&oxgLevel[j]);

     //Display the readings three values per line.
     for (j=0 ;j<6 ;j++)
     {
          if (j%3==0)
          printf("\n");
          printf("%d ",oxgLevel[j]);
     }
     }
```

In the program ,a series of oxygen levels is read by means of a loop in which the loop counter ,j , ranges from 0 to 5 .for each value of j, the statement

```
scanf("%d",&oxgLevel[j]);
```

read in an oxygen level and stores it in array element oxgLevel[j], for example if you entered the integer value

97 100 103 253 147 180

as the six oxygen level readings, your input data would be stored in oxgLevel as

oxgLevel[0]	97
oxgLevel[1]	100
oxgLevel[2]	103
oxgLevel[3]	253
oxgLevel[4]	147
oxgLevel[5]	180

Array oxgLevel can be initialized as

```
int oxgLevel[6]={97,100,103,253,147,180};
```

Characters Array or Strings:

A string constant is a one-dimensional array of characters terminated by a null (‘\0’). The terminator define end of string.

For example,

```
char name[7] = { 'H', 'A', 'S', 'E', 'E', 'B','\0' } ;

char name[7]= ”HASEEB”;
```

The above declaration and initialization create a string consisting of the word "HASEEB". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "HASEEB".

Index	0	1	2	3	4	5	6
Variable	H	A	S	E	E	B	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456	0x23457

The String I/O Function gets() & puts():

Sanf() and printf() is not versatile for string I/O we can use gets() and puts() function from stdio library.

For Example:

```
/*  Example Program For gets() and puts() In C */
```

```
#include<stdio.h>
void main(void)
{
	char name[20];
	printf("Enter a name for gets() :");

	//get string input using gets() function
	gets(name);

	printf("Entered name Is : will be with puts() :");
	//print string using puts() function
	puts(name);

}
```

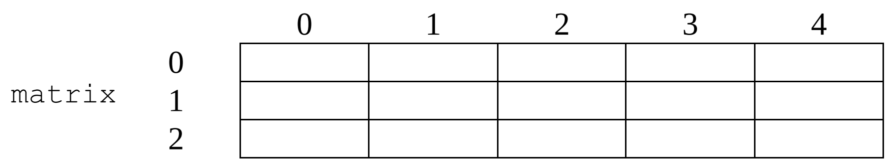
C supports a wide range of functions that manipulate null-terminated strings −

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

Two Dimensional Arrays:

In many programming applications you naturally organize data into rows and columns. In C you can use a two-dimensional array to store data in this form. The two dimensional arrays can be describe as “arrays of arrays”. All of them of a same uniform data type.

For example:

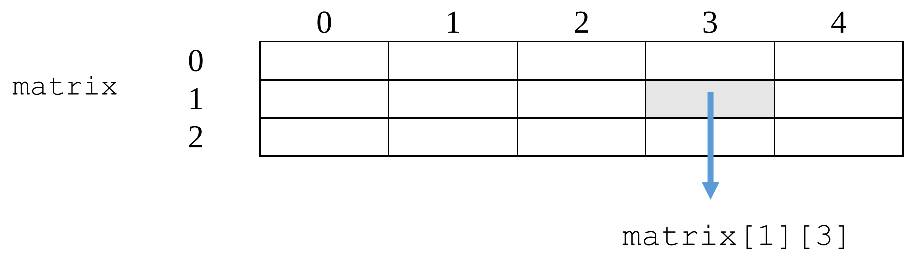


matrix represents a two-dimensional array of 3 per 5 elements of type int. The C syntax for this is:

```
 int matrix [3][5];
```

and, for example, the way to reference the second element vertically and fourth horizontally in an expression would be:

```
matrix[1][3]
```



```
//assign row*column integers in the array matrix

#define ROW 5
#define COLUMN 3

int matrix [ROW][COLUMN];
int n,m;

void main ()
{
  for (n=0; n<ROW; n++)
    for (m=0; m<COLUMN; m++)
    {
      matrix[n][m]=(n+1)*(m+1);
    }
}
```

		0	1	2	3	4
matrix	0	1	2	3	4	5
	1	6	7	8	9	10
	2	11	12	13	14	15

Or you can initialized the array as:

```
int matrix [ROW][COLUMN]={1,2,3,4,5
                      6, 7, 8,9,10,
                      11, 12, 13, 14, 15};
```

Arrays as parameters:

At some point, we may need to pass an array to a function as a parameter. In C, it is not possible to pass the entire block of memory represented by an array to a function directly as an argument. But what can be passed instead is its address (for arrays we don’t need to use & operator for address, we can access the address of first element by simply array name). In practice, this has almost the same effect, and it is a much faster and more efficient operation.

To accept an array as parameter for a function, the parameters can be declared as the array type, but with empty brackets, omitting the actual size of the array. For example:

```
void procedure (int arg[])
```

This function accepts a parameter of type "array of int" called arg. In order to pass to this function an array declared as:

```
int myarray [40];
```

it would be enough to write a call like this:

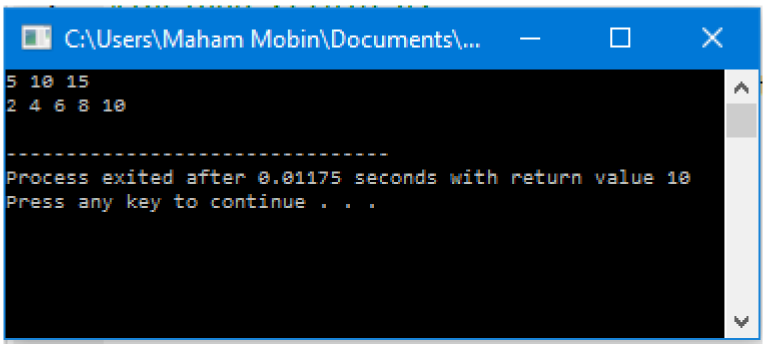
```
procedure (myarray);
```

Here you have a complete example:

```
// arrays as parameters
#include <stdio.h>

void printarray (int arg[], int length) {
int n;
for ( n=0; n<length; ++n)
    printf("%d ",arg[n]);
printf("\n");
}

void main ()
{
  int firstarray[3] = {5, 10, 15};
  int secondarray[5] = {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
}
```



```
5 10 15
2 4 6 8 10

--------------------------------
Process exited after 0.01175 seconds with return value 10
Press any key to continue . . .
```