CL1002

Programming Fundamentals

Lab 10

Recursion & Structures

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

LAB 10

Learning Objectives

This lab will cover the following topics:

- Recursion
- Structures

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.

Flowchart for recursion:

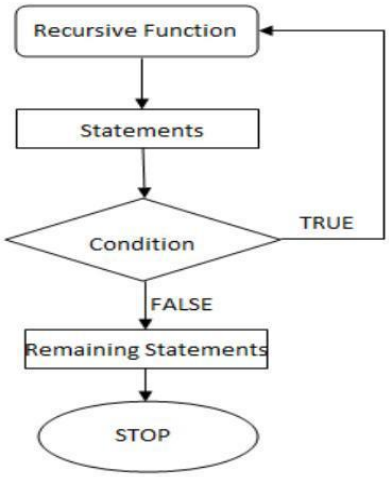


Fig: Flowchart showing recursion

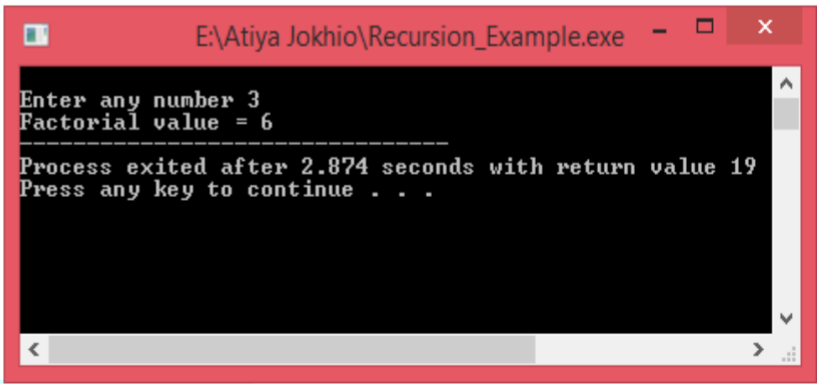
Each recursive solution has at least two cases: the base case and the general case.

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.

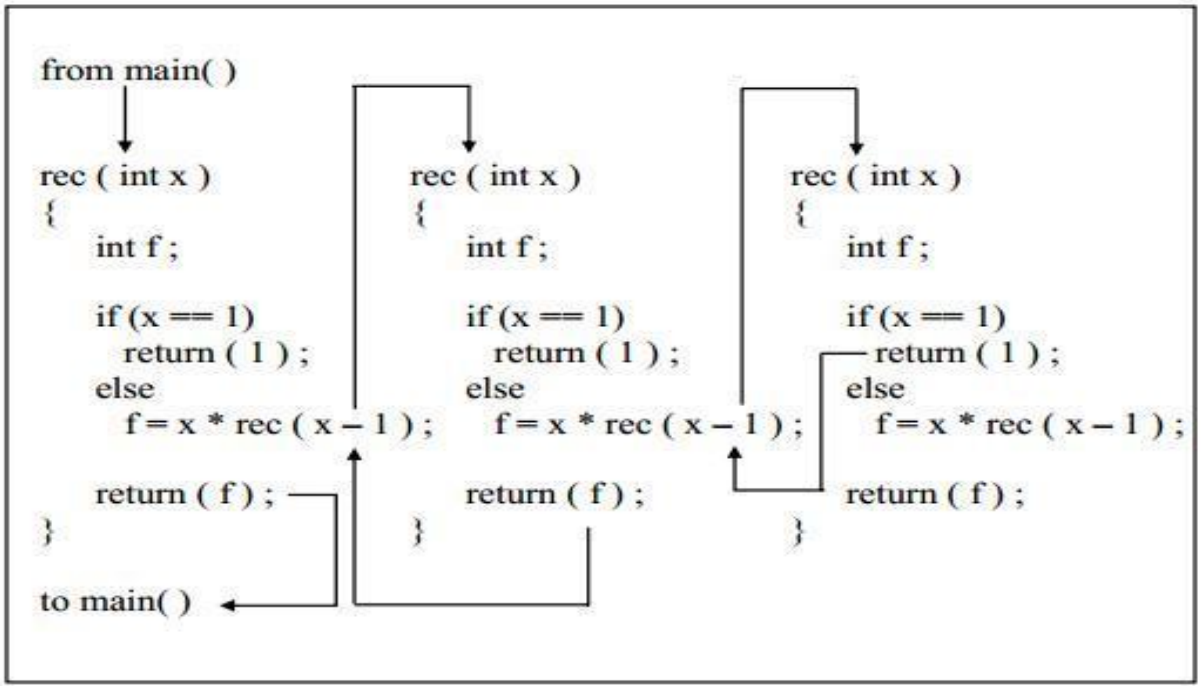
For example, a classic recursive problem is the factorial. The factorial of a number is defined as the number times the product of all the numbers between itself and 0: N! = N * (N-1)! The factorial of 0 is 1. We have a base case, Factorial (0) is 1, and we have a general case, Factorial (N) is N * Factorial (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.
Following is the recursive version of the function to calculate the factorial value.

```
#include <stdio.h>
int main( )
{
    int a, fact ;
    printf ( "\nEnter any number " ) ;
    scanf ( "%d", &a ) ;
    fact = rec ( a ) ;
    printf ( "Factorial value = %d", fact ) ;
    }
    rec ( int x )
    {
    int f ;
    if ( x == 1 )
    return ( 1 ) ;
    else
    f = x * rec ( x - 1 ) ;
    return ( f ) ;
    }
```



Assume that the number entered through scanf( ) is 3. The figure below explains what exactly happens when the recursive function rec( ) gets called.



Disadvantages of recursion

- Recursive programs are generally slower than non-recursive programs. This is because,recursive function needs to store the previous function call addresses for the correct
  Program jump to take place.
- Requires more memory to hold intermediate states. It is because, recursive programrequires the allocation of a new stack frame and each state needs to be placed into the stack frame, unlike non-recursive (iterative) programs.

Structures:

We studied earlier that array is a data structure whose element are all of the same data type. Now we are going towards structure, which is a data structure whose individual elements can differ in type. Thus a single structure might contain integer elements, floating– point elements and character elements. Pointers, arrays and other structures can also be included as elements within a structure. The individual structure elements are referred to as members. This lesson is concerned with the use of structure within a 'c' program. We will see how structures are defined, and how their individual members are accessed and processed within a program. The relationship between structures and pointers, arrays and functions will also be examined. Closely associated with the structure is the union, which also contains multiple members.

In general terms, the composition of a structure may be defined as

```
struct tag
{ member 1;
member 2;
------------
-------------
member m; }
```

In this declaration, struct is a required key-word; tag is a name that identifies structures of this type. The individual members can be ordinary variables, pointers, arrays or other structures. The member names within a particular structure must be distinct from one another, though a member name can be same as the name of a variable defined outside of the structure.

A storage class, however, cannot be assigned to an individual member, and individual members cannot be initialized within a structure-type declaration. For example:

```
struct student
{
char name [80];
introll_no;
```

```
float marks;
};
```

We can now declare the structure variable s1 and s2 as follows:

```
struct student s1, s2;
```

s1 and s2 are structure type variables whose composition is identified by the tag student.
It is possible to combine the declaration of the structure composition with that of the structure variable as shown below.

```
storage- class struct tag
{
member 1;
member 2;
  ——
  —-
  member m;
} variable 1, variable 2 --------- variable n;
```

The tag is optional in this situation.

```
struct student {
char name [80];
introll_no;
float marks;
} s1, s2;
```

The s1, s2, are structure variables of type student. Since the variable declarations are now combined with the declaration of the structure type, the tag need not be included. As a result, the above declaration can also be written as

```
struct {
char name [80];
introll_no;
float marks ;
} s1, s2;
```

A structure may be defined as a member of another structure. In such situations, the declaration of the embedded structure must appear before the declaration of the outer structure. The members of a structure variable can be assigned initial values in much the same manner as the elements of an array. The initial values must appear in the order in which they will be assigned

to their corresponding structure members, enclosed in braces and separated by commas. The general form is

```
storage-classstruct tag variable = { value1, value 2,-------,value m};
```

A structure variable, like an array can be initialized only if its storage class is either external or static. e.g. suppose there are one more structure other than student.

```
Struct dob
{
Int month;
int day;
int year; };

struct student
{ char name [80];
introll_no; float
marks;  structdob
d1; };

static struct student st = { "ali", 2, 99.9, 17, 11, 01};
```

PROCESSING A STRUCTURE

The members of a structure are usually processed individually, as separate entities. Therefore, we must be able to access the individual structure members. A structure member can be accessed by writing

variable.member name

This period (.) is an operator, it is a member of the highest precedence group, and its associativity is left-to-right.

E.g. if we want to print the detail of a member of a structure then we can write as printf("%s",st.name); or printf("%d", st.roll_no) and so on. Morecomplex expressions involving the repeated use of the period operator may also be written. For example, if a structure member is itself a structure, then a member of the embedded structure can be accessed by writing:

variable.member.submember

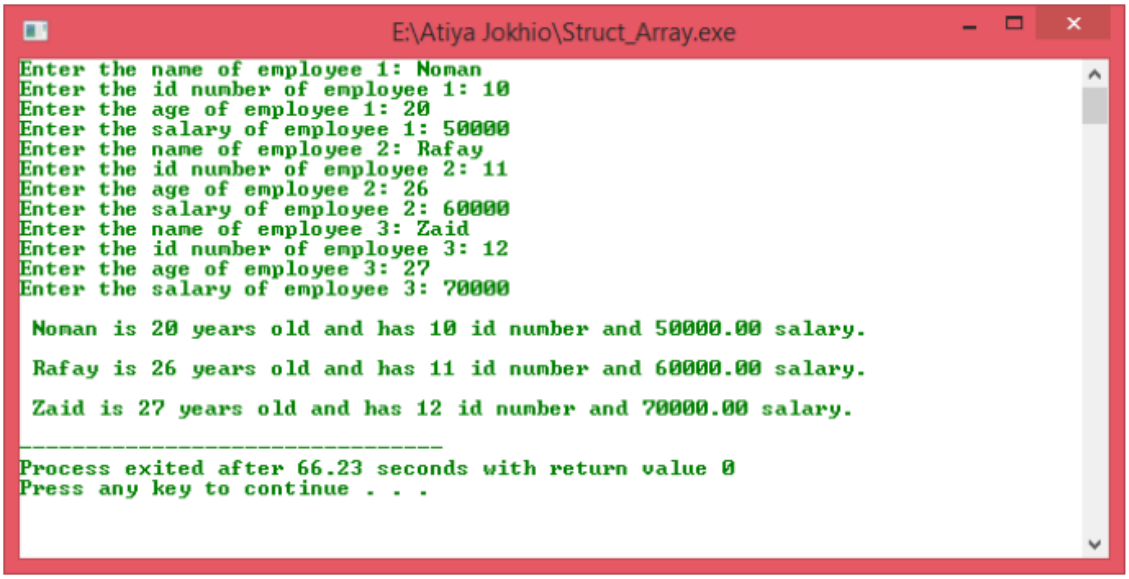
Thus in the case of student and dob structure, to access the month of date of birth of a student, we would write

st.d1.month

Array of Structures:

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types.

```c
#include<stdio.h>
//structure declaration
struct employee {
char name[25];
int id_number;
int age;
float salary;
};
int main()
{
system("color F2");
int i;
struct employee emp[3];  /* Structure array declaration */
for(i=0;i<3;i++)
{
printf("Enter the name of employee %d:", i+1 );
gets(emp[i].name);
printf("Enter the id number of employee %d:", i+1 );
scanf("%d",&emp[i].id_number);
printf("Enter the age of employee %d:", i+1);
scanf("%d",&emp[i].age);
printf("Enter the salary of employee %d:", i+1);
scanf("%f",&emp[i].salary);
fflush(stdin);   //clears the buffer
}
for(i=0;i<3;i++)
{
printf("\n%s is %d years old and has %d id number and %.2f salary.\n",
emp[i].name, emp[i].age,
emp[i].id_number,emp[i].salary);
}
```



Structures with function:

Structure variables are passed by value by default. To pass an array by value, create a structure with the array as a member.

```
#include<stdio.h>
#include<string.h>
int i;
//structure declaration
struct employee {
char name[25];
int id_number;
int age;
float salary;
};
void Input(struct employee emp[], int count)
{
for(i=0;i<count;i++)
{
printf("Enter the name of employee: " );
scanf("%s", emp[i].name);
printf("Enter the id number of employee: " );
scanf("%d",&emp[i].id_number);
printf("Enter the age of employee: " );
scanf("%d",&emp[i].age);
printf("Enter the salary of employee: " );
scanf("%f",&emp[i].salary);
}
Display(emp,count);
}
void Display(struct employee emp[],int count)
{
for(i=0;i<count;i++)
{
printf("\n%s is %d years old and has %d id number and %.2f salary.\n",emp[i].name, emp[i].age,
emp[i].id_number,emp[i].salary);
}
}
int main()
{
struct employee emp[3];  /* Structure array declaration */
Input(emp,3);
return 0;
}
```

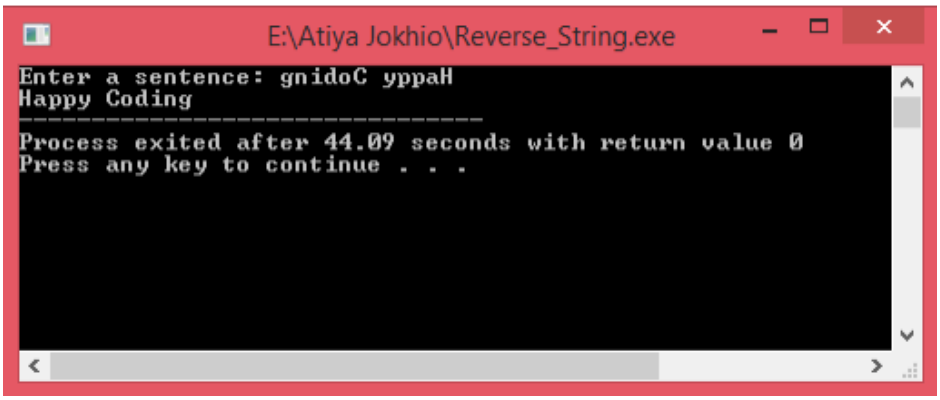
Lab Exercise

Question 1

Write a function that merges two sorted lists into a new sorted list. [1,4,6],[2,3,5] → [1,2,3,4,5,6]. You can do this quicker than concatenating them followed by a sort.

Question 2

Bit wants to send his message (Password of Account) to byte, in order to avoid that no one can understand his message he used the codewords. He only says to byte whatever will be written in given sentence, if you flip it right to left that will be its answer. Write a program in c (as displayed in output) using recursion.



Question 3

Write a program in C to multiply two matrix using recursion.

```
Enter the row and column of first matrix: 2
2
Enter the row and column of second matrix: 2
2
Enter the First matrix: 2
3
1
-2
Enter the Second matrix: 4
-5
8
3

The First matrix is: n
2 3
1 -2
The Second matrix is: n
4 -5
8 3
The multiplication of two matrixes is :

32 -1
-12 -11
--------------------------------
Process exited after 22.29 seconds with return value 0
Press any key to continue . . .
```

Question 4

Write a program in C to read and print its corresponding percentage from 1% to 100% using recursion.

Question 5:

A phone number, such as (212) 767-8900, can be thought of as having three parts: e.g., the area code (212), the exchange (767), and the number (8900). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone.

Create two structure variables of type phone. Initialize one, and have the user input a number for the other one. Then display both numbers.

The interchange might look like this:
Enter area code: 415
Enter exchange: 555
Enter number: 1212
Then display like below:
My number is (212) 767-8900
Your number is (415) 555-1212

Question 6:

Define a structure to store the following student data:

CGPA, courses (course name, GPA), address (consisting of street address, city, state, zip). Input 2 student records, compare and display which student have highest GPA in which course also Display which student has the highest CGPA .

HINT: define another structure to hold the courses and address.

Question 7:

Create a struct Rectangle with attributes length and width. Provide functions that calculate the perimeter and the area of the rectangle. The check() function should verify that length and width are each numbers larger than 0.0 and less than 20.0.

Question 8

Create a struct called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four data members—a part number (type string), a part description (type string), a quantity of the item being purchased (type int) and a price per item (type float). Your program should initialize the four data members. In addition, it should calculate the invoice amount (i.e., multiplies the quantity by the price per item), If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0. Write a test program that demonstrates struct Invoice's capabilities.