

CL1002
Programming
Fundamentals

LAB 10
Recursion, Structures and Nested
Structure

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Learning Objectives

1. Recursion
2. Structures
3. Nested Structures

1. 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:

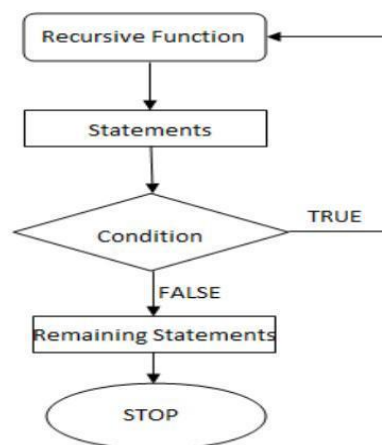


Fig: Flowchart showing recursion

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

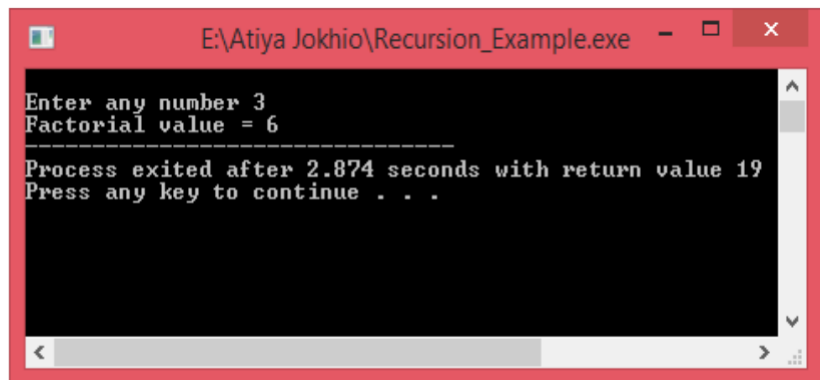
The **base case** is the one to which we have an answer; the **general case** expresses the solution in terms of a call to itself with a smaller version of the problem. Because the general case solves a smaller and smaller version of the original problem, eventually the program reaches the base case, 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 $N * \text{Factorial}(N-1)$. 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.

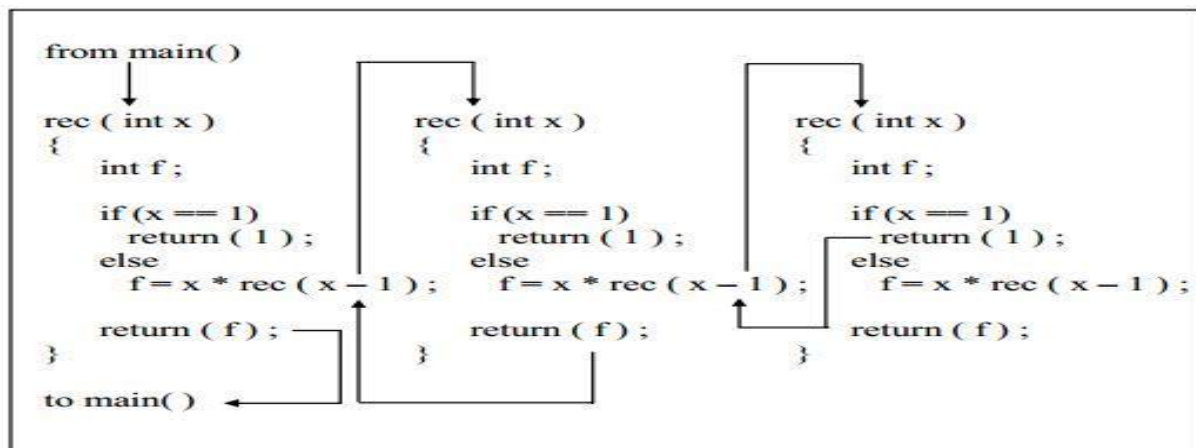
Following is the recursive version of the function to calculate the factorial value.

```
#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 program requires the allocation of a new stack frame and each state needs to be placed into the stack frame, unlike non-recursive (iterative) programs.

2. Structures

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly structure is another user defined data type available in C that allows to combine data items of different kinds. Structures are derived data types—they're constructed using objects of other types. Normally, we use structure to store the record or the details of any item or entity. Structure members can be variables of the primitive data types (e.g., int, float, etc.), or aggregates, such as arrays and other structures.

- Keyword struct introduces a structure definition
- The identifier Chocolate is the structure tag, which names the structure definition and is used with struct to declare variables of the structure type—e.g., struct Chocolate kitkat, Mars, Jubilee.
- Variables declared within the braces of the structure definition are the structure's members.
- Members of the same structure type must have unique names, but two different structure types may contain members of the same name without conflict.
- Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –
 - a. Title
 - b. Author
 - c. Subject
 - d. Book ID

2.1 Declaration of Struct

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.

```
struct Books {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int  book_id;  
};
```

```
struct Choclote{  
    char Name[20];  
    float Weight;  
    int Calories;  
    float Price;  
    char ExpiryDate[10];  
};
```

2.2 Declaration & Initialization of Struct type Variables

You can declare the variables before the semi-colon(;) or using a proper declaration syntax like other variable's in main(); To access any member of a structure, we use the member access operator (.). The

member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword struct to define variables of structure type. The following example shows how to use a structure in a program –

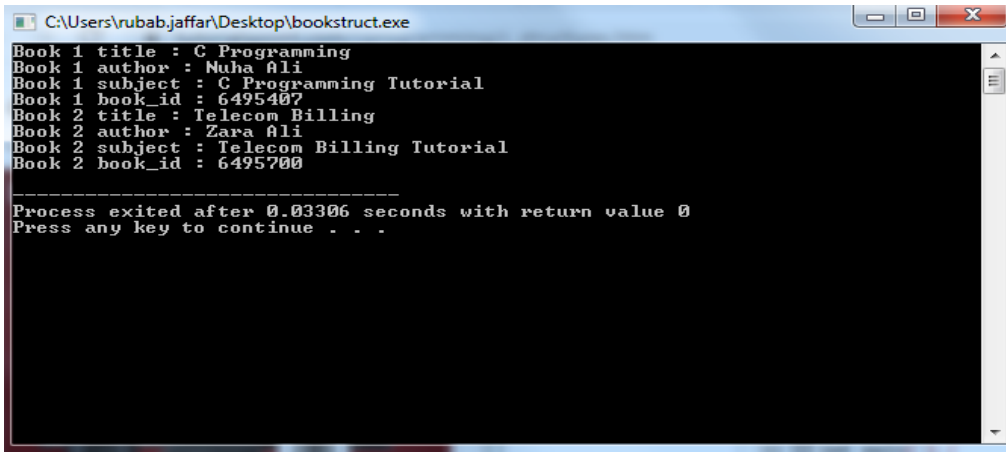
➤ **Before semi-colon**

```
struct Books {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
} Book1, Book2 ;
```

➤ **Proper declaration in Main function**



```
#include <stdio.h>  
#include <string.h>  
struct Books {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
};  
  
int main( ) {  
    struct Books Book1;    /* Declare Book1 of type Book */  
    struct Books Book2;    /* Declare Book2 of type Book */  
    /* book 1 specification */  
    strcpy( Book1.title, "C Programming");  
    strcpy( Book1.author, "Nuha Ali");  
    strcpy( Book1.subject, "C Programming Tutorial");  
    Book1.book_id = 6495407;  
  
    /* book 2 specification */  
    strcpy( Book2.title, "Telecom Billing");  
    strcpy( Book2.author, "Zara Ali");  
    strcpy( Book2.subject, "Telecom Billing Tutorial");  
    Book2.book_id = 6495700;  
    /* print Book1 info */  
    printf( "Book 1 title : %s\n", Book1.title);  
    printf( "Book 1 author : %s\n", Book1.author);  
    printf( "Book 1 subject : %s\n", Book1.subject);  
    printf( "Book 1 book_id : %d\n", Book1.book_id);  
    /* print Book2 info */  
    printf( "Book 2 title : %s\n", Book2.title);  
    printf( "Book 2 author : %s\n", Book2.author);  
    printf( "Book 2 subject : %s\n", Book2.subject);  
    printf( "Book 2 book_id : %d\n", Book2.book_id);  
    return 0;  
}
```



```
C:\Users\rubab.jaffar\Desktop\bookstruct.exe
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700

-----
Process exited after 0.03306 seconds with return value 0
Press any key to continue . . .
```

2.3 Declaration & Initialization of Struct type Array

```
#include <<stdio.h>>
#define MAX_STUDENTS 5
struct student
{
    char name[100];
    int roll;
    float marks;
};
int main()
{
    struct student stu[MAX_STUDENTS];
    int i;
    printf("Enter %d student details\n", MAX_STUDENTS);
    for ( i = 0; i < MAX_STUDENTS; i++ )
    {
        printf("Student %d name: ", (i + 1));
        gets(stu[i].name);
        printf("Student %d roll no: ", (i + 1));
        scanf("%d", &stu[i].roll);
        printf("Student %d marks: ", (i + 1));
        scanf("%f", &stu[i].marks);
        getchar(); // <-- Eat extra new line character
        printf("\n");
    }
    // Print all student details
    printf("\n\nStudent details\n");
    printf("-----\n");
    for ( i = 0; i < MAX_STUDENTS; i++ )
    {
        printf("Name : %s\n", stu[i].name);
        printf("Roll : %d\n", stu[i].roll);
        printf("Marks: %.2f\n", stu[i].marks);
        printf("-----\n");
    }
    return 0;
}
```

2.4 Nested Structures

Nested structure in C is nothing but structure within structure. One structure can be declared inside another structure as we declare structure members inside a structure. The structure variables can be a normal structure variable, array or a pointer variable to access the data.

As a Separate structure:

```
#include <stdio.h>
#include <string.h>

struct UniversityDetails
{
    int UniversityRanking;
    char UniversityName[90];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct UniversityDetails data;
};

int main()
{
    struct student_detail std_data = {1, "Arif", 80.5, 285,
                                      "National University of Computer & Emerging Sciences"};
    printf(" Id is: %d \n", std_data.id);
    printf(" Name is: %s \n", std_data.name);
    printf(" Percentage is: %f \n\n", std_data.percentage);

    printf(" University Ranking is: %d \n",
           std_data.data.UniversityRanking);
    printf(" University Name is: %s \n",
           std_data.data.UniversityName);
    return 0;
}
```

OUTPUT:

```
Id is: 1
Name is: Arif
Percentage is: 80.500000

University Ranking is: 285
University Name is: National University of Computer & Emerging Sciences

-----
Process exited after 0.1215 seconds with return value 0
Press any key to continue . . .
```

As an Embedded structure:

```
nestedst.cpp ×
1  #include <stdio.h>
2  #include <string.h>
3
4  struct Outer {
5      int a;
6      int b;
7      struct Inner {
8          int c;
9      } in;
10 };
11
12 int main() {
13     struct Outer out = {5, 10, {15}};
14     printf("Outer variable a = %d \n", out.a);
15     printf("Outer variable b = %d \n", out.b);
16     printf("Inner variable c = %d \n", out.in.c);
17 }
```


Another example of Nested Structure:

Sample Code:

```
#include <stdio.h>
#include <string.h>

struct Type{
    char TypeName[20];    // Mini, Sedan, Sports, Luxury, SUV
};

struct Car{
    char CarName[20];
    char make[15];
    char model[15];
    char color[10];
    int seats;
    int engine;    // 1800 cc
    int price;

    struct Type CarType;
};

int main()
{
    struct Car myCar;

    puts("----- Example: Nested Structure -----");

    puts("Enter the Name of your Car: ");
    gets(myCar.CarName);
    puts("Enter the type of your Car {Mini, Sedan, Sports, Luxury, SUV}: ");
    gets(myCar.CarType.TypeName);

    puts("Enter the Color of your Car: ");
    gets(myCar.color);
    puts("Enter the make of your Car: ");
    gets(myCar.make);
    puts("Enter the model of your Car: ");
    gets(myCar.model);
    printf("\nEnter the seats of your Car: ");
    scanf("%d",&myCar.seats);
    printf("\nEnter the engine capacity (cc) of your Car: ");
    scanf("%d",&myCar.engine);
    printf("\nEnter the price of your Car: ");
    scanf("%d",&myCar.price);

    puts("\n\n----- Print -----");

    printf("\nCarName: %s",myCar.CarName);
    printf("\nCarType: %s",myCar.CarType.TypeName);
    printf("\nColor: %s",myCar.color);
    printf("\nMake: %s",myCar.make);
    printf("\nModel: %s",myCar.model);
    printf("\nSeats: %d",myCar.seats);
    printf("\nEngine (cc): %d",myCar.engine);
    printf("\nPrice: %d", myCar.price);

    return 0;
}
```

OUTPUT:

```
----- Example: Nested Structure -----
Enter the Name of your Car:
Picanto 2.0
Enter the type of your Car {Mini, Sedan, Sports, Luxary, SUV}:
Mini
Enter the Color of your Car:
White
Enter the make of your Car:
KIA
Enter the model of your Car:
Picanto

Enter the seats of your Car: 4

Enter the engine cpacity (cc) of your Car: 1300

Enter the price of your Car: 120000

----- Print -----

CarName: Picanto 2.0
CarType: Mini
Color: White
Make: KIA
Model: Picanto
Seats: 4
Engine (cc): 1300
Price: 120000
-----
Process exited after 54.59 seconds with return value 0
Press any key to continue . . .
```