SR
INSTITUTE OF SCIENCE ANDTECHNOLOGY,
CHENNAI.

18CSS101J – Programming for ProblemSolving

Unit III

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

UNIT III

Initializing and accessing of 2D Array – Initializing multidimensional Array – Array programs 2D – Array contiguous memory – Array advantages and Limitations – Array construction for real-time application common programming errors – String Basics – String Declaration and Initialization – String Functions: gets(), puts(), getchar(), putchar(), printf() - String Functions: atoi, strlen, strcat strcmp – String Functions: sprintf, sscanf, strrev, strcpy, strstr, strtok – Arithmetic characters on strings.

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

UNIT III

- Functions declaration and definition : Types: Call by Value, Call by Reference – Function with and without Arguments and no Return values –

Functions with and without Arguments and Return Values – Passing Array to function with return type – Recursion Function

3.1INITIALIZING AND ACCESSING OF 2D ARRAY

3.1.1 ACCESSING OF 2D ARRAY

- Two Dimensional Array requires ***Two Subscript*** Variables
- Two Dimensional Array stores the values in the form of matrix.
- One Subscript Variable denotes the "***Row***" of a matrix.
- Another Subscript Variable denotes the "***Column***" of a matrix.

3.1.2 INITIALIZING OF 2D ARRAY

An array of two dimensions can be declared as follows:

data_type array_name[size1][size2];

Here ***data_type*** is the name of some type of data, such as int. Also, ***size1*** and ***size2*** are sizes of the array's first and second dimensions respectively.

Declaration and Use of Two Dimensional Array :

int a[3][4];

Use :

```
for(i=0;i<row,i++)
   for(j=0;j<col,j++)
   {
   printf("%d",a[i][j]);
}
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Meaning of Two Dimensional Array :

- Matrix is having 3 rows (i takes value from 0 to 2)
- Matrix is having 4 Columns (j takes value from 0 to 3)
- Above Matrix 3×4 matrix will have 12 blocks having 3 rows & 4 columns.
- Name of 2-D array is 'a' and each block is identified by the row & column number.
- Row number and Column Number Starts from 0.
- If a[0][0] means 0^{th} row 0^{th} column, a[0][1] means 0^{th} row 1^{st} column, a[0][2] means 0^{th} row 2^{nd} column like wise it goes on…….

	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]

Initializing 2D array:

- Row wise assignment
- Combine assignment
- Selective assignment

Method 1 : INITIALIZING ALL ELEMENTS ROWWISE

For initializing 2D Array we can need to assign values to each element of an array using the below syntax.

```
int a[3][2] = {
        { 1 , 4 },
        { 5 , 2 },
        { 6 , 5 }
        };
```

Consider the below program –

```
#include<stdio.h>
int main() {
  int i, j;
  int a[3][2] = { { 1, 4 },
                  { 5, 2 },
                  { 6, 5 }};
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 2; j++) {
      printf("%d ", a[i][j]);
    }
    printf("\n");
  } return 0;
}
```

Output :

1 4
5 2
6 5

We have declared an array of size 3 X 2, It contain overall 6 elements. Row 1 : { 1 , 4 },

Row 2 : { 5 , 2 },

Row 3 : { 6 , 5 }

We have initialized each row independently

a[0][0] = 1

A[0][1] = 4

Method 2 : COMBINE ASSIGNMENT

Initialize all Array elements but initialization is much straight forward. All values are assigned sequentially and row-wise

```
int a[3][2] = {1 , 4 , 5 , 2 , 6 , 5 };
```

So here it automatically assigns that number 3 has row and number 2 has column.

Consider the below program –

```
#include <stdio.h>
int main() {
   int i, j;
   int a[3][2] = { 1, 4, 5, 2, 6, 5 };
   for (i = 0; i < 3; i++) {
      for (j = 0; j < 2; j++) {
         printf("%d ", a[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

Output :

1 4
5 2
6 5

Method 3: SELECTIVE ASSIGNMENT

```
int a[3][2] = {
    { 1 },
    { 5 , 2 },
    { 6 }
};
```

Now we have again going with the way 1 but we are removing some of the elements from the array. In this case we have declared and initialized 2-D array like this

Consider the below program –

```
#include <stdio.h>
int main() {
  int i, j;
  int a[3][2] = { { 1 },
                  { 5, 2 },
                  { 6 }};
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 2; j++) {
      printf("%d ", a[i][j]);
    } printf("\n"); }
  return 0;
}
```

Output :

1 0
5 2
6 0

3.2 INITIALIZING MULTIDIMENSIONAL ARRAY

Arrays with more than one dimension are called multidimensional arrays.

Declaration of three-Dimensional arrays:

data_type array_name[size1][size2][size3];

Arrays do not have a shaped like squares and cubes; each dimension of the array be given in different size as follows:
Eg: int non_cube[2][[6][8];

You can initialize a three dimensional array in a similar way like a two dimensional array. Here's an example,

```
int test[2][3][4] = {
                    { {3, 4, 2, 3}, {0, -3, 9, 11}, {9, 1, 6, 2} },
                    { {13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9} }
                    };
```

In this program we mention size1 – row, size2 – column, size3 – number of elements in that array.

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

C Program to store values entered by the user in a three-dimensional array and display it.

```
#include <stdio.h>

int main()

{
   int i, j, k, test[2][3][2]; // this array can store 12 elements

    printf("Enter 12 values: \n");

    for(i = 0; i < 2; ++i) {

        for (j = 0; j < 3; ++j) {

            for(k = 0; k < 2; ++k ) {

                scanf("%d",
```

```
}
}
printf("\nDisplaying values:\n");//Displaying values with
                                        proper   index.
   for(i = 0; i < 2; ++i) {
      for (j = 0; j < 3; ++j) {
         for(k = 0; k < 2; ++k )
         {
            printf("test[%d][%d][%d] = %d\n", i, j, k,
            test[i][j][k]);
                  }
         }
}
   return 0;
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

OUTPUT:

Enter 12 values:

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

Displaying Values:

test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12

3.3 ARRAY PROGRAMS –
2D

a) Basic 2d array program

b) Store and display values

c) Sum of two matrices using Two dimensional arrays

d) Transpose of Matrix

e) Multiplication of 2d matrices.

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Basic 2d array program

```
#include<stdio.h>
int  main(){
     int disp[2][3]; /* 2D
array  declaration*/
     int i, j; /*Counter
     variables
for loop*/
   for(i=0; i<2; i++) {
     for(j=0;j<3;j++) {
        printf("Enter value for
disp[%d][%d]:", i, j);
        scanf("%d", &disp[i][j]);
     } }
```

```
//Displaying array elements
   printf("Two Dimensional
array  elements:\n");
   for(i=0; i<2; i++) {
     for(j=0;j<3;j++) {
           printf("%d ",
            disp[i][j]);  if(j==2){
printf("\n");
        }
                                    }    }
   return 0; }
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Basic 2d array program

OUTPUT

Enter value for disp[0][0]:1 Enter value for disp[0][1]:2 Enter value for disp[0][2]:3 Enter value for disp[1][0]:4 Enter value for disp[1][1]:5 Enter value for disp[1][2]:6

Two Dimensional array elements: 1 2 3

4 5 6

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

STORE AND DISPLAY VALUES OF ARRAY

```
#include <stdio.h>
const int CITY = 1;
const int WEEK =
7;  int main()
{
   int
   temperature[CITY][WEEK];
   for (int i = 0; i < CITY; ++i) {
      for(int j = 0; j < WEEK; ++j) {
         printf("City %d, Day %d: ",
i+1, j+1);
         scanf("%d",
&temperature[i][j])
;
```

```
printf("\nDisplaying    values:
\n\n");
   for (int i = 0; i < CITY; ++i) {
      for(int j = 0; j < WEEK;
      ++j)
      {
%d\n",i+1,j+1,
temperature[i][j])
;
      }
   }
   return 0;
}
```

printf("City %d, Day %d =

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

OUTPUT

```
City 1, Day 1: 33
City 1, Day 2: 34
City 1, Day 3: 35
City 1, Day 4: 33
City 1, Day 5: 32
City 1, Day 6: 31
City 1, Day 7: 30
```

```
Displaying values:
City 1, Day 1 = 33
City 1, Day 2 = 34
City 1, Day 3 = 35
City 1, Day 4 = 33
City 1, Day 5 = 32
City 1, Day 6 = 31
City 1, Day 7 = 30
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

SUM OF 2D ARRAY

```
#include <stdio.h>
int main()
{
  float a[2][2], b[2][2], c[2][2];
  int i, j;  // Taking input using nested
for loop
  printf("Enter elements of 1st
matrix\n");
  for(i=0; i<2; ++i)
   for(j=0; j<2; ++j)
   {
     printf("Enter a%d%d: ", i+1, j+1);
     scanf("%f", &a[i][j]);
   } // Taking input using nested for
   loop
  printf("Enter elements of 2nd
matrix\n");
  for(i=0; i<2; ++i)
   for(j=0; j<2; ++j)
   {
     printf("Enter b%d%d: ", i+1, j+1);
     scanf("%f", &b[i][j]);
   }// adding corresponding elements of
two arrays
  for(i=0; i<2; ++i)
   for(j=0; j<2; ++j)
   {
     c[i][j] = a[i][j] + b[i][j];
   }
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

```
// Displaying the sum
   printf("\nSum Of Matrix:");
   for(i=0; i<2; ++i)
    for(j=0; j<2; ++j)
    {
       printf("%.1f\t", c[i][j]);
            if(j==1)
          printf("\n");
    }
return 0;
}
```

OUTPUT:

Enter elements of 1st matrix
Enter a11: 2;
Enter a12: 0.5;
Enter a21: -1.1;
Enter a22: 2;
Enter elements of 2nd matrix
Enter b11: 0.2;
Enter b12: 0;
Enter b21: 0.23;
Enter b22: 23;
Sum Of Matrix:
2.2 0.5
-0.9 25.0

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

TRANSPOSE OF MATRIX

```
#include <stdio.h>
int main()
{
   int a[10][10], transpose[10][10], r, c,
i, j;
   printf("Enter rows and columns
   of
matrix:");
   scanf("%d %d", &r, &c); //
Storing elements
   printf("\nEnter elements of
matrix:\n");
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
       printf("Enter element a%d%d:
",i+1, j+1);
         scanf("%d", &a[i][j]);  }
   printf("\nEntered Matrix: \n");
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
         printf("%d ", a[i][j]);
         if (j == c-1)
            printf("\n\n");
      }// Finding the transpose
matrix a of
   for(i=0; i<r; ++i)
      for(j=0; j<c; ++j)
      {
         transpose[j][i] = a[i][j];
      }
```

```
// Displaying the transpose of matrix
   a  printf("\nTranspose of
   Matrix:\n");  for(i=0; i<c; ++i)
      for(j=0; j<r; ++j)
      {
         printf("%d ",transpose[i][j]);
         if(j==r-1)
            printf("\n\n");
      }
   return 0;
}
```

output

```
Enter rows and columns of matrix: 2 3
Enter element of matrix:
Enter element a11: 2
Enter element a12: 3
Enter element a13: 4
Enter element a21: 5
Enter element a22: 6
Enter element a23: 4
Entered Matrix:
2     3 4
5     6 4
Transpose of
Matrix:  2    5
3     6
4     4
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

MULTIPLCATION OF 2D MATRIX

```
#include
<stdio.h>  int
main()
{ int m, n, p, q, c, d, k, sum = 0;
  intfirst[10][10],     second[10][10],
multiply[10][10];
   printf("Enter number of rows and
columns of first matrix\n");
scanf("%d%d", &m, &n);
printf("Enter    elements    of    first
matrix\n");
   for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)
      scanf("%d", &first[c][d]);
 printf("Enter     number of     rows
      and  columns of second
matrix\n");  scanf("%d%d", &p, &q);
 for (c = 0; c < p; c++)
     for (d = 0; d < q; d++)
       scanf("%d", &second[c][d]);
    for (c = 0; c < m; c++) {
     for (d = 0; d < q; d++) {
     for (k = 0; k < p; k++) {
        sum     =     sum+
first[c][k]*second[k][d];
      }
       multiply[c][d] = sum;
      sum = 0;
    } }
```

```
 printf("Product    of   the
matrices:\n");
    for (c = 0; c < m; c++)
     {  for (d = 0; d < q;
     d++)
       printf("%d\t", multiply[c][d]);
      printf("\n");
   }   }
}return 0;
```

OUTPUT

Enter the number of rows and columns of first matrix: 33
Enter the elements of first matrix: 1 2 0
0 1 1
2 0 1
Enter the number of rows and columns of second matrix:
3
3
Enter elements of the second matrix:
1 1 2
2 1 1
1 2 1
Product of entered matrices:-

5	3	4
3	3	2
3	4	5

4. ARRAY CONTIGUOUS MEMORY

- When Big Block of memory is reserved or allocated then that memory block is called as Contiguous Memory Block.
- Alternate meaning of Contiguous Memory is continuous memory.
- Suppose inside memory we have reserved 1000-1200 memory addresses for special purposes then we can say that these 200 blocks are going to reserve contiguous memory.

How to allocate contiguous memory:

- Using static array declaration.
- Using alloc() / malloc() function to allocate big chunk of memory dynamically.

Contiguous Memory Allocation

- Two registers are used while implementing the contiguous memory scheme. These registers are base register and limit register.

When OS is executing a process inside the main memory then
content of each register are as –

- Register - Content of register.
- Base register- Starting address of the memory location where process execution is happening.
- Limit register- Total amount of memory in bytes consumed by process.

When process try to refer a part of the memory then it will firstly refer the base address from base register and then it will refer relative address of memory location with respect to base address

3.5 ADVANTAGE AND LIMITATIONS OF ARRAY

Advantages:

- It is better and convenient way of storing the data of same datatype with same size.
- It allows us to store known number of elements in it.
- It allocates memory in contiguous memory locations for its elements. It does not allocate any extra space/ memory for its elements. Hence there is no memory overflow or shortage of memory in arrays.
- Iterating the arrays using their index is faster compared to any other methods like linked list etc.
- It allows to store the elements in any dimensional array - supports multidimensional array.

3.5.2 Limitations of Array:

A) Static Data

- Array is Static data Structure
- Memory Allocated during Compile time.
- Once Memory is allocated at Compile Time it Cannot be Changed during Run-time.

B) Can hold data belonging to same Data types

- Elements belonging to different data types cannot be stored in array because array data structure can hold data belonging to same data type.
- Example : Character and Integer values can be stored inside separate array but cannot be stored in single array

C) Inserting data in Array is Difficult

- Inserting element is very difficult because before inserting element in an array have to create empty space by shifting other elements one position ahead.
- This operation is faster if the array size is smaller, but same operation will be more and more time consuming and non-efficient in case of array with large size.

D) Deletion Operation is difficult

- Deletion is not easy because the elements are stored in contiguous memory location.
- Like insertion operation , we have to delete element from the array and after deletion empty space will be created and thus we need to fill the space by moving elements up in the array.

E) Bound Checking

- If we specify the size of array as 'N' then we can access elements upto 'N-1' but in C if we try to access elements after 'N-1' i.e Nth element or N+1th element then we does not get any error message.
- Process of Checking the extreme limit of array is called *Bound checking* and C does not perform Bound Checking.
- If the array range exceeds then we will get garbage value as result.

E) Shortage of Memory

- Array is Static data structure. Memory can be allocated at *compile time* only Thus if after executing program we need more space for storing additional information then we cannot allocate additional space at run time.
- Shortage of Memory , if we don't know the size of memory in advance

F) Wastage of Memory

- Wastage of Memory , if array of large size is defined.

3.6 ARRAY CONSTRUCTION FOR REAL-TIME APPLICATION COMMON PROGRAMMING ERRORS

(i) Constant Expression Require

```
#include<stdio.h
>  void main()
{
int i=10;
int a[i];
}
```

In this example we see what's that error?

- We are going to declare an array whose size is equal to the value of variable.
- If we changed the value of variable then array size is going to change.
- According to array concept, we are allocating memory for array at compile time so if the size of array is going to vary then how it is possible to allocate memory to an array.
- *i is initialized to 10 and using a[i] does not mean a[10] because 'i' is Integer Variable whose value can be changed inside program.*

- **Value of Const Variable Cannot be changed**
- we know that value of Const Variable cannot be changed once initialized so we can write above example as below –

```
#include<stdio.h
> void main()
{
const int
i=10; int a[i];
}
```

or

int a[10];

(ii) Empty Valued 1D Array

```
#include<stdio.h
> void main()
{
int arr[];
}
```

Instead of it Write it as –

```
#include<stdio.h
> void main()
{
int a[] = {1,1};
}
```

- Consider this example, we can see the empty pair of square brackets means we haven't specified size of 1D array. In this example array is empty. undefined or
- Size of 1D Array should be Specified as a Constant Value.

```
#include<stdio.h>
void main()
{
int a[] = {};// This Cause also an Error
```

(iii) 1D Array with no Bound Checking

```
#include<stdio.h>
void main()
{
int a[5];
printf("%d",a[7]);
}
```

Here Array size specified is 5.

- So we have Accessto Following Arra Elements a[0],a[1],a[2],a[3] and a[4].
- But accessing a[5] causes Garbage Value to be used because C Does not performs Array Bound Check.

If the maximum size of array is “MAX” then we can access following elements of an
array –
Elements accessible for Array Size "MAX" = arr[0]
=.
= arr[MAX-1]

4. Case Sensitive

```
#include<stdio.h
>  void main()
{
int a[5];
printf("%d",A[2]);
}
```

Array Variable is Case Sensitive so A[2] does not print anything it Displays Error Message : “Undefined Symbol A“

3.7 STRING BASICS

- **Strings in C are represented by arrays of characters.**
- String is nothing but the collection of the individual array elements or characters stored at contiguous memory locations

  i) Character array – 'P','P','S'

  ii) Double quotes - “PPS" is a example of String.
  - If string contains the double quote as part of string then we can use escape character to keep double quote as a part of string.

iii) Null Character

- The end of the string is marked with a special character, the *null character*, which is a character all of whose bits are zero i.e., a NULL.
- String always Terminated with NULL Character (‘/0’)

char name[10] = {'P','P','S','\0'}

  - NULL Character is having ASCII value 0
  - ASCII Value of '\0' = 0
  - As String is nothing but an array , so it is Possible to Access Individual Character

name[10] = "PPS";

- It is possible to access individual character

name[0] = 'P';
name[1] = 'P';
name[2] = 'S';
name[3] = '\0';

iv) MEMORY

Each Character Occupy 1 byte of Memory

Size of "PPS" = Size of 'P' +

= Size of 'P' +

= Size of 'S'

; Size of "PPS" is 3 BYTES

Each Character is stored in consecutive memory location.

Address of 'P' = 2000 Address of 'P' = 2001 Address of 'S' = 2002

8. STRING DECLARATION AND INITIALIZATION

- ### 3.8.1 String Declaration:
  - String data type is not supported in C Programming. String means Collection of Characters to form particular word. String is useful whenever we accept name of the person, Address of the person, some descriptive information. We cannot declare string using String Data Type, instead of we use array of type character to create String.
  - Character Array is Called as 'String'.
  - Character Array is Declared Before Using it in Program.

char String_Variable_name [SIZE] ;

Eg: char city[30];

Point	Explanation
Significance	- We have declared array of character[i.e String]
Size of string	- 30 Bytes
Bound checking	- C Does not Support Bound Checking i.e if we store City with size greater than 30 then C will not give you any error
Data type	- char
Maximum size	- 30

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Precautions to be taken while declaring Character Variable :

- String / Character Array Variable name should be legal C Identifier.
- String Variable must have Size specified.
  - **char city[];**
- Above Statement will cause compile time error.
  - Do not use String as data type because String data type is included in later languages such as C++ / Java. C does not support String data type
  - **String city;**
- When you are using string for other purpose than accepting and printing data then you must include following header file in your code –
  - **#include<string.h>**

2. *Initializing String [Character Array] :*

- Whenever we declare a String then it will contain garbage values inside it. We have to initialize String or Character array before using it. Process of Assigning some legal default data to String is Called Initialization of String. There are different ways of initializing String in C Programming –

  1)Initializing Unsized Array of Character

  2)Initializing String Directly

  3)Initializing String Using Character Pointer

Way 1 : Unsized Array and Character

- ***Unsized Array*** : Array Length is not specified while initializing character array using this approach
- Array length is Automatically calculated by Compiler
- Individual Characters are written inside Single Quotes , Separated by comma to form a list of characters. Complete list is wrapped inside Pair of Curly braces
- ***NULL Character*** should be written in the list because it is ending or terminating character in the String/Character Array
- char name [] = {'P','P','S','\0'};

Way 2 : Directly initialize String Variable

- In this method we are directly assigning String to variable by writing text in double quotes.
- In this type of initialization , we don’t need to put NULL or Ending / Terminating character at the end of string. It is appended automatically by the compiler.
- char name [ ] = "PPS";

Way 3 : Character Pointer Variable

- Declare Character variable of pointer type so that it can hold the base address of “String”
- Base address means address of first array element i.e (address of name[0])
- NULL Character is appended Automatically
- char *name = "PPS";

9. STRING FUNCTIONS

- gets()
- puts()
- getchar()
- putchar()
- printf()
- atoi()
- strlen()
- strcat ()
- strcmp()
- sprintf()
- sscanf()
- strrev()
- strcpy()
- strstr()
- strtok()

3.9.1 gets():

Syntax for Accepting String :

char * gets (char * str); OR

gets(<variable-name>)

Example :

```
#include<stdio.h>
void main()
{
char name[20];
printf("\nEnter the Name : ");
gets(name);
}
```

Output:

Enter the name: programming in c

Explanation :

- Whenever gets() statement encounters then characters entered by user (the string with spaces) will be copied into the variable.
- If user start accepting characters , and if new line character appears then the newline character will not be copied into the string variable(i.e name).
- A terminating null character is automatically appended after the characters copied to string vriable (i.e name)
- gets() uses stdin (Standered Input Output) as source, but it does not include the ending newline character in the resulting string and does not allow to specify a maximum size for string variable (which can lead to buffer overflows).

Some Rules and Facts :

A. %s is not Required :

Like scanf statement %s is not necessary while accepting string. scanf("%s",name);

and here is gets() syntax which is simpler than scanf() –

gets(name);

B. Spaces are allowed in gets() :

gets(name);

Whenever the above line encounters then interrupt will wait for user to enter some text on the screen. When user starts typing the characters then all characters will be copied to string and when user enters newline character then process of accepting string will be stopped.

Sample Input Accepted by Above Statement :

- Value Accepted : Problem solving\n
- Value Stored : Problem solving (\n Neglected)

3.9.2 puts():

Way 1 :Messaging

- puts(" Type your Message / Instruction ");
- Like Printf Statement puts() can be used to display message.

Way 2 : Display String

- puts(string_Variable_name) ;

Notes or Facts :

- puts is included in header file “stdio.h”
- As name suggest it used for Printing or Displaying Messages or Instructions.

Example :

```
#include<  stdio.h>
#include< conio.h>
void main()
  {
   char string[] = "This is an example
   string(\n";
   puts(string);     // String is variable Here
   puts("String");   // String is in Double Quotes
   getch();
}
```

Output :

String is : This is an example

string String is : String

3.9.3 getchar():

- getchar() function is also one of the function which is used to accept the single character from the user.
- The characters accepted by getchar() are buffered until RETURN is hit means getchar() does not see the characters until the user presses return. (i.e Enter Key)

Syntax for Accepting String and Working :

/* getchar accepts character & stores in ch */

```
char ch = getchar();
```

- When control is on above line then getchar() function will accept the single character. After accepting character control remains on the same line. When user presses the enter key then getchar() function will read the character and that character is assigned to the variable ‘ch’.

ParameterExplanation

Header File	-	stdio.h
Return Type	-	int (ASCII Value of the character)
Parameter	-	Void
Use	-	Accepting the Character

Example 1 :

In the following example we are just accepting the single character and printing it on the console –

```
main()
{
   char ch;
   ch = getchar();
   printf("Accepted Character : %c",ch);
}
```

Output :

Accepted Character : A

Example 2 : Accepting String (Use One of the Loop)

```
#include<stdio.h>
void main()
{
int i = 0;
char name[20];
printf("\nEnter the Name : ");
while((name[i] = getchar())!='\n')
      i++ ;/*while loop will accept the one character at a time and check it with newline character. Whenever user enters newline character then control comes out of the loop.*/
getch();
}
```

3.9.4 putchar():

Displaying String in C Programming

Syntax :

```
putchar(int c);
```

Way 1 : Taking Character as Parameter `putchar('a') ;` // Displays :

a

- Individual Character is Given as parameter to this function.
- We have to explicitly mention Character.

Way 2 : Taking Variable as Parameter

```
putchar(a);
```

// Display Character Stored in a

- Input Parameter is Variable of Type “Character”.
- This type of putchar() displays character stored in variable.

Way3:Displaying Particular Character from Array

```
putchar(a[0]) ;
```

// Display a[0] th element from array

- Character Array or String consists of collection of characters.
- Like accessing individual array element , characters can be displayed one by one using putchar().

Example:

```
#include<  stdio.h>
#include< conio.h>
int main()
{
   char string[] = "C
   programming\n";  int i=0;
   while(string[i]!='\0')
   {
     putchar(string[i])
     ;  i++;
   }
return 0;
}
```

Output:

C programming

3.9.5 printf():

Syntax :

Way 1 : Messaging

```
printf (" Type your Message / Instruction " ) ;
```

Way 2 : Display String

```
printf ("Name of Person is %s ", name ) ;
```

Notes or Facts :

printf is included in header file "**stdio.h**"

As name suggest it used for **Printing or Displaying Messages or Instructions Uses :**

- Printing Message
- Ask user for entering the data (Labels . Instructions)
- Printing Results

3.10.1 atoi() Function:

- atoi = A to I = Alphabet to Integer
- **Convert String of number into Integer**

Syntax:

num = atoi(String);

num - Integer Variable
String- String of Numbers

Example :

```
char a[10] = "100";
int value = atoi(a);
printf("Value = %d\n",
value);  return 0;
```

Output :
Value : 100

Significance :

- Can Convert any String of Number into Integer Value that can Perform the arithmetic Operations like integer
- Header File : **stdlib.h**

Ways of Using Atoi Function :

Way 1 : Passing Variable in Atoi Function

```
int num;
char marks[3] = "98";
num = atoi(marks);
printf("\nMarks : %d",num);
```

Way 2 : Passing Direct String in Atoi Function

```
int num;
num = atoi("98");
printf("\nMarks : %d",num);
```

OTHER INBUILT TYPECAST FUNCTIONS IN C PROGRAMMING LANGUAGE:

- Typecasting functions in C language performs data type conversion from one type to another.
- Click on each function name below for description and example programs.

  - atof() Converts string to float
  - atoi() Converts string to int
  - atol() Converts string to long
  - itoa() Converts int to string
  - ltoa() Converts long to string

3.10.2 strlen() FUNCTION:

- **Finding length of string**

Point	Explanation
No of Parameters	- 1
Parameter Taken	- Character Array Or String
Return Type	- Integer
Description	- Compute the Length of the String
Header file	- string.h

Different Ways of Using strlen() :

- There are different ways of using strlen function. We can pass different parameters to strlen() function.

Way 1 : Taking String Variable as Parameter

```
char str[20];
int length ;
printf("\nEnter the String :
"); gets(str);
length = strlen(str);
printf("\nLength of String : %d ",
length);
```

Output:

```
Enter the String :
hello  Length of String
: 5
```

Way 2 : Taking String Variable which is Already Initialized using Pointer

```
char *str = "priteshtaral";
int length ;
length = strlen(str);
printf("\nLength of String : %d ", length);
```

Way 3 : Taking Direct String

```
int length ;
length = strlen("pritesh");
printf("\nLength of String :
%d",length);
```

Way 4 : Writing Function in printf Statement

```
char *str = "pritesh";
printf("\nLength of String : %d", strlen(str));
```

3.10.3 strcat() FUNCTION:

What strcat Actually does ?

- Function takes 2 Strings / Character Array as Parameter
- Appends second string at the end of First String.

Parameter Taken - 2 Character Arrays / Strings

Return Type - Character Array / String

Syntax :

char* strlen (char * s1, char * s2);

Ways of Using Strcat Function :

Way 1 : Taking String Variable as Parameter

```
char str1[20] = “Don” , str2[20] =
“Bosqo”;  strcat(str1,str2);
puts(str1);
```

Way 2 : Taking String Variable which is Already Initialized using Pointer

```
char *str1 = “Ind”,*str2 = “ia”;
strcat(str1,str2); // Result stored in str1
puts(str1);    //Result:  India
```

Way 3 : Writing Function in printf Statement

```
printf(“String:     “,
strcat(“Ind”,”ia”));
```

3.10.4 strcmp() FUNCTION:

What strcmp Actually Does ?

- Function takes two Strings as parameter.
- It returns integer .

Syntax :

```
int strcmp ( char *s1, char *s2 ) ;
```

Return Type		**Condition**
-ve Value	-	String1 < String2
+ve Value	-	String1 > String2
0 Value	-	String1 = String2

Example 1 : Two strings are Equal

```
char s1[10] = "SAM",s2[10]="SAM"
;
int len;
len = strcmp
(s1,s2);  Output
0
/*So  the output will be  0.  if  u  want  to  print  the string
then  give  condition like*/
char s1[10] = "SAM",s2[10]="SAM" ;
int len;
len = strcmp
(s1,s2);  if (len == 0)
printf ("Two Strings are Equal");
```

Output:

Example 2 : String1 is Greater than String2

```
char s1[10] = "SAM",s2[10]="sam"
;  int len;
len = strcmp (s1,s2);
printf ("%d",len); //-ve value
```

Output:

-32

Reason :

ASCII value of “SAM” is smaller than “sam”

ASCII value of ‘S’ is smaller than ‘s’

Example 3 : String1 is Smaller than String1

```
char s1[10] = "sam",s2[10]="SAM"
;  int len;
len = strcmp (s1,s2);
printf ("%d",len); //+ve value
```

Output:

85

Reason :

ASCII value of “SAM” is greater than “sam”

ASCII value of ‘S’ is greater than ‘s’

3.11.1 sprintf() FUNCTION:

- **sends formatted output to String**.

Features :

- Output is Written into String insteadof Displaying it on the Output Devices.
- Return value is integer (i.e Number of characters actually placed in array / length of string).
- String is terminated by '\0'.
- Main Per pose : Sending Formatted output to String.
- Header File : **Stdio.h**

Syntax :

int sprintf(char *buf,char format,arg_list);

Example :

```
int age = 23 ;
char str[100];
sprintf( str , "My age is
%d",age);  puts(str);
```

Output:

My age is 23

Analysis of Source Code: Just keep in mind that

- Assume that we are using printf then we get output “My age is 23”
- What does printf does ? —– Just Print the Result on the Screen
- Similarly Sprintf stores result “My age is 23” into string str instead of printing it.

3.11.2 sscanf() FUNCTION:

Syntax :

int sscanf(const char *buffer, const char *format[, address, ...]);

What it actually does ?

- Data is read from array Pointed to by buffer rather than stdin.
- Return Type is Integer
- Return value is nothing but number of fields that were actually assigned a value

Example

```
#include <stdio.h>
int main ()
{
  char buffer[30]="Fresh2refresh 5
  ";  char name [20];
  int age;
  sscanf (buffer,"%s %d",name,&age);
  printf ("Name : %s \n Age : %d
  \n",name,age);  return 0;
}
```

Output:

Name : Fresh2refresh Age : 5

10.11.3 strstr() FUNCTION:

- **Finds first occurrence of sub-string in other string**

Features :

- Finds the first occurrence of a sub string in another string
- Main Purpose : **Finding Substring**
- Header File : **String.h**
- Checks whether s2 is present in s1 or not
- On success, strstr returns a pointer to the element in s1 where s2 begins (points to s2 in s1).
- On error (if s2 does not occur in s1), strstr returns null

Syntax :

char *strstr(const char *s1, const char *s2);

```
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a test string";
  char *p;
  p = strstr (string,"test");
  if (p)
  {
      printf("string found\n");
```

```
   printf("First string \"test\" in \"%s\" to "\" \"%s \"" ,string, p);
  }
  else
          printf("string not found\n" );
   return 0;
}
```

Output:

string found

First string “test” in “This is a test string” to “test string”.

Example 2

Parameters as String initialized using Pointers

```
#include<stdio.h>
#include<string.h>
int main()
{
  char *str1 = "c4learn.blogspot.com", *str2 = "spot",
  *ptr;  ptr = strstr(str1, str2);
  printf("The substring is: %sn",
  ptr);  return 0;
}
```

Output:

The substring is: spot.com

Example 3
Passing Direct Strings

```
#include<stdio.h>
#include<string.h>
void main()
{
  char *ptr;
  ptr =
  strstr("c4learn.blogspot.com","spot");
  printf("The substring is: %sn", ptr);
}
```

Output:

The substring is: spot.com

3.11.4 strrev() Function:

- reverses a given string in C language. Syntax for strrev( ) function is given below.

char *strrev(char *string);

- strrev() function is nonstandard function which may not available in standard library in C.

Algorithm to Reverse String in C :

- Start
- Take 2 Subscript Variables ‘i’,’j’
- ‘j’ is Positioned on Last Character
- ‘i’ is positioned on first character
- str[i] is interchanged with str[j]
- Increment ‘i’
- Decrement ‘j’
- If ‘i’ > ‘j’ then goto step 3
- Stop

Example

```
#include<stdio.h>
#include<string.h>
int main()
{
    char name[30] = "Hello";
        printf("String before strrev() :%s\n",name);
        printf("String after strrev()%s", strrev(name));
        return 0;
}
```

Output:

```
String before strrev() : Hello
String after strrev()     : olleH
```

3.11.5 strcpy() FUNCTION:

Copy second string into First

What strcmp Actually Does ?

- Function takes two Strings as parameter.
- Header File : String.h.
- It returns string.
- Purpose : Copies String2 into String1.
- Original contents of String1 will be lost.
- Original contents of String2 will remains as it is.

Syntax :

char * strcpy (char *string1, char *string2)

;

- strcpy (str1, str2) – It copies contents of str2 into str1.
- strcpy (str2, str1) – It copies contents of str1 into str2.
- If destination string length is **less than source string**, **entire source string value won't be copied into destination** string.
- For example, consider destination string length is 20 and source string length is 30. Then, only 20 characters from source string will be copied into destination string and remaining 10

Example 1

```
char s1[10] = "SAM" ;

char s2[10] = "MIKE" ;

strcpy (s1,s2);

puts (s1) ;     // Prints : MIKE
puts (s2) ;     // Prints : MIKE
```

Output:

MIKE

MIKE

Example 2

```
#include <stdio.h>
#include
<string.h>  int
main( )
{
  char source[ ] = "hihello" ;
  char target[20]= "" ;
  printf ( "\nsource string = %s", source )
  ;  printf ( "\ntarget string = %s", target )
  ;  strcpy ( target, source ) ;
 printf("target string after strcpy()=%s",target)
   ;  return 0; }
```

Output

```
source string = hihello
target string =
target string after strcpy( ) = hihello
```

3.11.6 strtok() FUNCTION:

- **tokenizes/parses the given string using delimiter.**

Syntax

char * strtok (char * str, const char * delimiters);

For example, we have a comma separated list of items from a file and we want individual items in an array.

- *Splits str[] according to given delimiters and returns next token.*
- *It needs to be called in a loop to get all tokens.*
- *It returns NULL when there are no more tokens.*

Example

```
#include <stdio.h>

#include <string.h>

int main()

{
        char str[] = "Problem_Solving_in_c";//Returns first token

        char* token = strtok(str, "_");//Keep printing tokens while
one of the delimiters present in

        str[].  while (token != NULL) {

                printf("%s\n", token);
```

```
    token = strtok(str, "_");
        }
        return 0;
}
```

Output:

Problem Solving in C

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

3.12 ARITHMETIC CHARACTERS ON STRING

- C Programming Allows you to Manipulate on String
- Whenever the Character is variable is used in the expression then it is automatically Converted into Integer Value called ASCII value.
- All Characters can be Manipulated with that Integer Value.(Addition,Subtraction)

Examples :

ASCII value of : ‘a’ is 97

Possible Ways of Manipulation

.

Way 1:Displays ASCII value[Note that %d in Printf]

```
char x = 'a';

printf("%d",x); // Display Result = 97
```

Way 2 :Displays Character value[Note that %c in Printf]

```
char x = 'a';

printf("%c",x); // Display Result = a
```

Way 3 : Displays Next ASCII value[Note that %d in Printf]

```
char x = 'a' + 1 ;

printf("%d",x); //Display Result = 98 (ascii of 'b' )
```

Way 4 Displays Next Character value[Note that %c in Printf]

```
char x = 'a' + 1;

printf("%c",x); // Display Result = 'b‘
```

Way 5 : Displays Difference between 2 ASCII in Integer[Note %d in Printf] char x = 'z' - 'a';

```
printf("%d",x);/*Display Result = 25 (difference between ASCII of z and a )
*/
```

Way 6 : Displays Difference between 2 ASCII in Char [Note that %c in Printf]

```
char x = 'z' - 'a';

printf("%c",x);/*Display Result =( difference between ASCII of z and a ) */
```

13. FUNCTION DECLARATION AND DEFINITION:

- A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.
- You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.
- A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.
- The C standard library provides numerous built-in functions that your program can call. For example, **strcat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.
- A function can also be referred as a method or a sub-routine or a procedure, etc.

3.13.1 Defining a function

The general form of a function definition in C programming language

is as follows −

```
return_type function_name( parameter list )
{
body of the function
}
```

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function −

Return Type − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Function Name – This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body – The function body contains a collection of statements that define what the function does

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

```
main()
{
display();
}
void mumbai()
{
printf("In mumbai");
}
void pune()
{
india();
}
void display()
{
pune();
}
void india()
{
mumbai();
}
```

We have written functions in the above specified sequence , however functions are called in which order we call them.

Here **functions are called this sequence** –

main()
display()
pune()
india()
mumbai().

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Why Funtion is used???

Advantages of Writing Function in C Programming

1. **Modular and Structural Programming can be done**
   - ❑ We can divide c program in **smaller modules**.
   - ❑ We can call module whenever require. e.g suppose we have written calculator program then we can write 4 modules (i.e add,sub,multiply,divide)
   - ❑ Modular programming **makes C program more readable**.
   - ❑ Modules once created , **can be re-used in other programs**.

2. **It follows Top-Down Execution approach , So main can be kept very small.**
   - ❑ Every C program starts **from main function**.
   - ❑ Every function is **called directly or indirectly through main**
   - ❑ Example : **Top down approach**. (functions are executed from top to bottom)

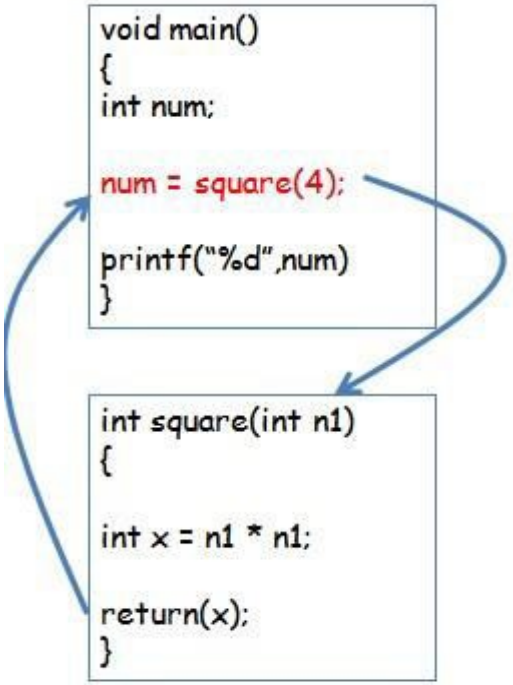
3. **Individual functions can be easily built,tested**
   - ❑ As we have developed C application in modules **we can test each and every module**.
   - ❑ **Unit testing** is possible.
   - ❑ Writing code in function will **enhance application development process**.
3. **Program development become easy**
4. **Frequently used functions can be put together in the customized library**
   - ❑ We can put frequently used functions in **our custom header file**.
   - ❑After creating header file we can re use header file. We can include header file in other program.
5. **A function can call other functions & also itself**
   - ❑ Function can call other function.
   - ❑ Function can call itself , which is called as “recursive” function.
   - ❑ Recursive functions are also useful in order to write system functions.
6. **It is easier to understand the Program topic**
   - ❑ We can get overall idea of the project just by reviewing function names.

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

R

How Function works in C Programming?

- C programming is modular programming language.
- We must divide C program in the different modules in order to create more readable, eye catching ,effective, optimized code.
- In this article we are going to see how function is C programming works ?



Explanation : How function works in C Programming ?

- Firstly Operating System will call our main function.
- When control comes inside main function , execution of main starts (i.e execution of C program starts)
- Consider Line 4 :

  num = square(4);
- We have called a function square(4). [See : How to call a function ?].
- We have passed “4” as parameter to function.

Note : Calling a function halts execution of the current function , it will execute called function

- after execution control returned back to the calling function.
- Function will return 16 to the calling function.(i.e. main)
- Returned value will be copied into variable.
- printf will gets executed.
- main function ends.
- C program terminates.

FUNCTION PROTOTYPE DECLARATION IN C PROGRAMMING

- Function prototype declaration is necessary in order to provide information the compiler about function, about return type, parameter list and function name etc.

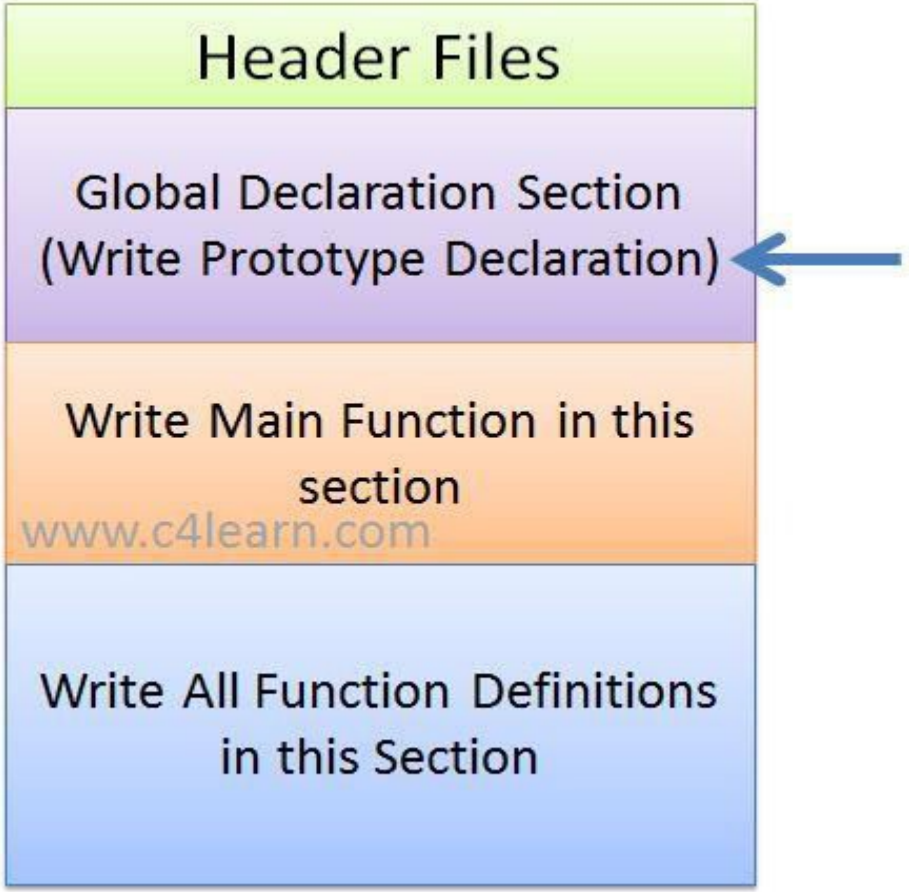
Important Points :

- Our program starts from main function. Each and every function is called directly or indirectly through main function
  - Like variable we also need to declare function before using it in program.
  - In C, declaration of function is called as prototype declaration
  - Function declaration is also called as function prototype

Points to remember

- Below are some of the important notable things related to prototype declaration –
- It tells name of function,return type of function and argument list related information to the compiler
  - Prototype declaration always ends with semicolon.
  - Parameter list is optional.
  - Default return type is integer.

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.



Syntax

return_type function_name (type arg1, type arg2.....);

prototype declaration comprised of three parts i.e name of the function,return type and parameter list

Examples of prototype declaration

- Function with two integer arguments and integer as return type is represented using syntax **int sum(int,int);**
- Function with integer argument and integer as return type is represented using syntax **int square(int);**
- In the below example we have written function with no argument and no return type **void display(void);**
- In below example we have delclared function with no argument and integer as return type **int getValue(void);**

Positioning function declaration

- If function definition is written after main then and then only we write prototype declaration in global declaration section
- If function definition is written above the main function then ,no need to write prototype declaration

Case 1 : Function definition written before main

```
#include<stdio.h>
void displayMessage()
{
printf(“welcome");
}
void main()
{
displayMessage();
}
```

Case 2 : Function definition written after main

```
#include<stdio.h> //Prototype Declaration
void displayMessage();
void main()
{
displayMessage();
}
void displayMessage()
{
printf(“welcome");
}
```

Need of prototype declaration

- Program Execution always starts from main , but during lexical analysis (1st Phase of Compiler) token generation starts from left to right and from top to bottom.
- During code generation phase of compiler it may face issue of backward reference.

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

14. TYPES OF CALLING

- While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.
- When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
- To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example –

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

```
#include <stdio.h>
/* function declaration */
int max(int num1, int num2);
int main ()
{
/* local variable definition */
 int a = 100;
 int b = 200;
 int ret;    /* calling a function to get max value
 */  ret = max(a, b);
printf( "Max value is : %d\n", ret );
return 0;
} /* function returning the max between two numbers */
int max(int num1, int num2)
{      /* local variable declaration
*/  int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result; }
```

output

Max value is : 200

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

We have kept max() along with main() and compiled the source code. While running the final executable, it would produce the following result –
Max value is : 200

- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.
- Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.
- While calling a function, there are two ways in which arguments can be passed to a function –
- By default, C uses **call by value** to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

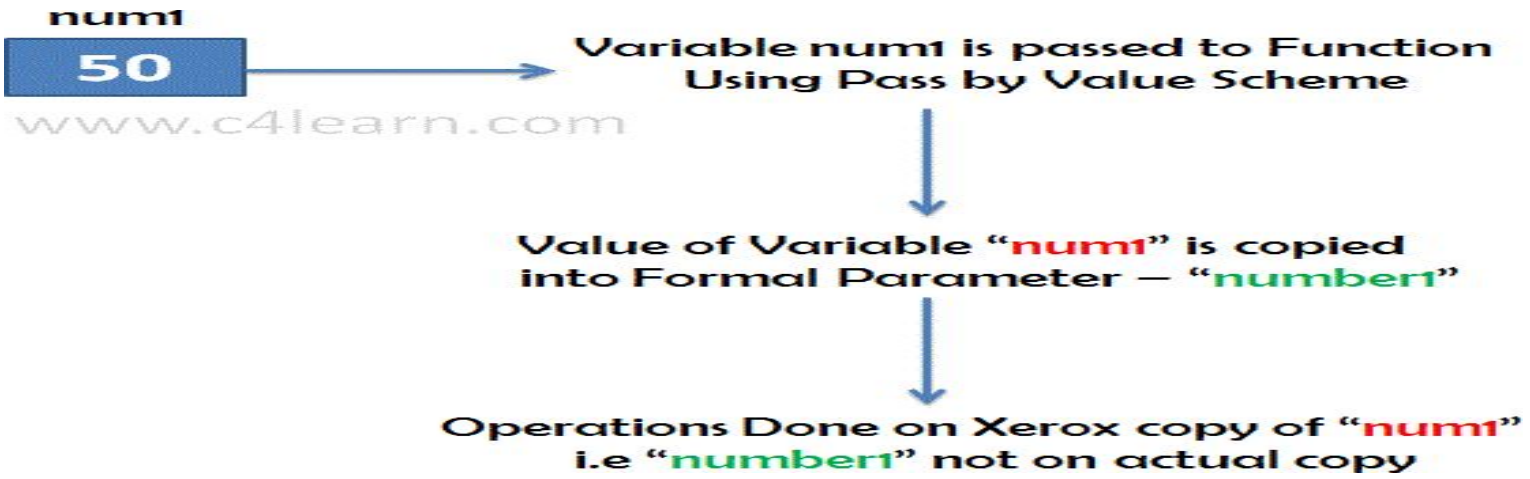
Sr.No.	Call Type & Description

1	**Call by value** This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
	/* function definition to swap the values
	*/ void swap (int x, int y)
	{
	int temp; temp = x;
	/* save the value of x */
	x = y; /* put y into x */
	y = temp; /* put temp into y
	*/ return; }

2	**Call by reference** This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.
	/* function definition to swap the values */
	void swap (int *x, int *y)
	{
	int temp;
	temp =
	*x;
	*x = *y;
	*y = temp;
	return; }

1. CALL BY VALUE

- While Passing Parameters using call by value , xerox copy of original parameter is created and passed to the called function.
- Any update made inside method will not affect the original value of variable in calling function.
- In the above example num1 and num2 are the original values and xerox copy of these values is passed to the function and these values are copied into number1,number2 variable of sum function respectively.
- As their scope is limited to only function so they cannot alter the values inside main function.



```
#include <stdio.h>
void swap(int x, int y); /* function declaration
*/  int main ()
{
int a = 100; /* local variable definition */
int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );/*calling a function to swap the values
*/  swap(a, b);
printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
return 0;
}
```

Output:

Before swap, value of a :100

Before swap, value of b

:200 After swap, value of a

:100 After swap, value of b

:200

2. CALL BY REFERENCE

- The **call by reference** method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.

- To pass a value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following which function **swap()**, exchanges the values of the two integer their variables pointed to, by arguments.

```
#include <stdio.h>
void swap(int *x, int *y); /* function declaration */
int main ()
{
int a = 100; /* local variable definition */
int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b ); /*calling a function to swap the values. * &a indicates pointer to a ie. address of variable a and * &b indicates pointer to b ie. address of variable b.*/
swap(&a, &b);
printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
return 0;
}
```

output

Before swap, value of a :100

Before swap, value of b

:200 After swap, value of a

:200 After swap, value of b

:100

15. .1 FUNCTION WITH ARGUMENTS AND NO RETURN VALUE :

- Function accepts argument but it does not return a value back to the calling Program .
- It is Single (One-way) Type Communication
- Generally Output is printed in the Called function

Function declaration : void function();

Function call : function();

Function definition : void function() { statements; }

```
#include <stdio.h>
void checkPrimeAndDisplay(int n);
int main()
{
int n; printf("Enter a positive integer:
");  scanf("%d",&n);
// n is passed to the function checkPrimeAndDisplay(n);
return 0;
}
```

```
// void indicates that no value is returned from the function
void checkPrimeAndDisplay(int n)
{
int i, flag = 0;
for(i=2; i <= n/2; ++i)
{
if(n%i == 0){
flag = 1;
break; }
}
if(flag == 1)
printf("%d is not a prime
number.",n);  else
printf("%d is a prime number.", n);
}
```

OUTPUT

Enter a positive integer :

4 4 is not a prime

number

3.15.2 FUNCTION WITH NO ARGUMENTS AND NO RETURN VALUE IN C

When a function has no arguments, it does not receive any data from the calling function. Similarly when it does not return a value, the calling function does not receive any data from the called function.

Syntax :

Function declaration : void function();

Function call : function();

Function definition : void function()

{

statements;

}

```
#include<stdio.h>
void area(); // Prototype Declaration
void main()
{
area();
}
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

```
void area()
{
   float area_circle;
   float rad;
    printf("\nEnter the radius : ");
   scanf("%f",&rad);
    area_circle = 3.14 * rad * rad ;
    printf("Area of Circle =
    %f",area_circle);
}
```

Output :

Enter the radius : 3

Area of Circle = 28.260000

3.16.1 FUNCTION WITHOUT ARGUMENTS AND RETURN VALUE

There could be occasions where we may need to design functions that may not take any arguments but returns a value to the calling function. A example for this is getchar function it has no parameters but it returns an integer an integer type data that represents a character.

Function declaration : int function();

Function call : function();

Function definition : int function() { statements; return x; }

```
#include<stdio.h>
 int sum();
 int main()
{
    int addition;
    addition = sum();
    printf("\nSum of two given values = %d", addition);
```

```
 return 0;
}

int sum()
{
   int a = 50, b = 80, sum;
   sum = a + b;
   return sum;
}
```

OUTPUT

Sum of two given values = 130

3.16.2 FUNCTION WITH ARGUMENTS AND RETURN VALUE

Syntax :

Function declaration : int function (int);

Function call : function(x);

Function definition: int function(int x) { statements; return x; }

```
#include<stdio.h>

float calculate_area(int);

int main()

 {

int radius;

float area;
printf("\nEnter the radius of the circle : ");

scanf("%d",&radius);

area = calculate_area(radius);

printf("\nArea of Circle : %f ",area);
```

```
return(0);
}
float calculate_area(int radius)
{
float areaOfCircle;
areaOfCircle = 3.14 * radius * radius;
return(areaOfCircle);
}
```

Output:

Enter the radius of the circle :

2 Area of Circle : 12.56

17. PASSING ARRAY TO FUNCTION IN C

Array Definition :

Array is collection of elements of similar data types .

Passing array to function :

Array can be passed to function by two ways :

- **Pass Entire array**
- **Pass Array element by element**

1 . Pass Entire array

- Here entire array can be passed as a argument to function .
- Function gets **complete access** to the original array .
- While passing entire array Address of first element is passed to function , any changes made inside function , directly **affects the Original value** .
- Function Passing method : **"Pass by Address"**

2 . Pass Array element by element:

- Here individual elements are passed to function as argument.
- Duplicate **carbon copy of Original variable** is passed to function .
- So any changes made inside function **does not affects the original value.**
- Function doesn't get complete access to the original array element.
- Function passing method is **"Pass by Value".**

Passing entire array to function :

- Parameter Passing Scheme : **Pass by Reference**
- Pass name of array as function parameter .
- Name contains the base address i.e (Address of 0th element)
- Array values are updated in function .
- Values are reflected inside main function also.

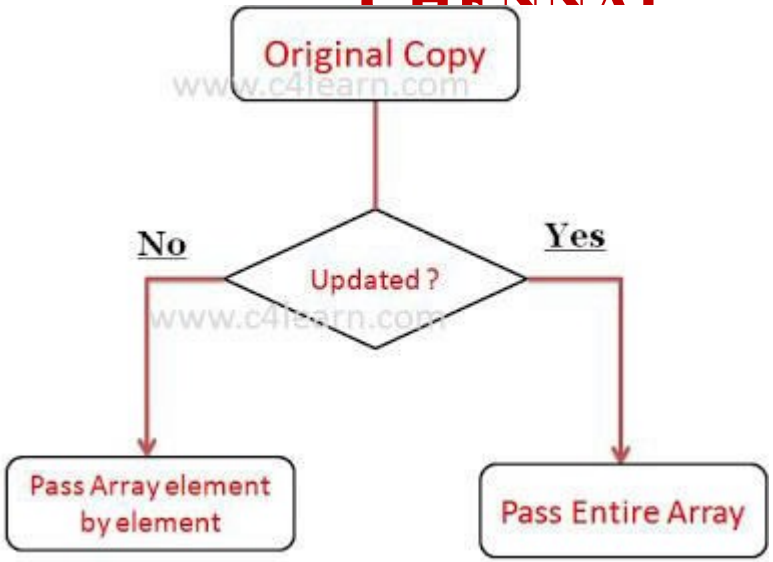

```
#include<stdio.h>

#include<conio.h>

void fun(int arr[])

{

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

arr[i] = arr[i] + 10;

}

void main()

{

int arr[5],i;

clrscr();

printf("\nEnter the array elements : ");
```

```
for(i=0;i< 5;i++)
scanf("%d",&arr[i]);

printf("\nPassing entire array

.....");


fun(arr); // Pass only name of array

for(i=0;i< 5;i++)
printf("\nAfter Function call a[%d] :

%d",i,arr[i]);  getch();


}
```

output

```
Enter the array elements : 1 2 3 4 5
Passing entire array .....

After Function call a[0] : 11

After Function call a[1] : 12

After Function call a[2] : 13

After Function call a[3] : 14
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI



Passing Entire 1-D Array to Function in C Programming

- Array is passed to function Completely.
- Parameter Passing Method : **Pass by Reference**
- It is Also Called "**Pass by Address**"
- Original Copy is Passed to Function
- Function Body Can Modify **Original Value.**

Example :

```
#include<stdio.h>
#include<conio.h>
void modify(int b[3]);
void main()
{
int arr[3] = {1,2,3};
modify(arr);
for(i=0;i<3;i++)
printf("%d",arr[i]);
getch();
}
void modify(int a[3])
{
int i;
for(i=0;i<3;i++)
a[i] = a[i]*a[i];
 }
```

Output :

1 4 9

Here **"arr"** is same as **"a"** because Base Address of Array **"arr"** is stored in Array **"a"**

Alternate Way of Writing Function Header :

Passing array element by element to function :

- ❑Individual element is passed to function using **Pass By Value** parameter passing scheme
- ❑Original Array elements remains same as Actual Element is never Passed to Function. thus function body cannot modify **Original Value.**
- ❑Suppose we have declared an array 'arr[5]' then its individual elements are arr[0],arr[1]…arr[4]. Thus we need 5 function calls to pass complete array to a function.

Tabular Explanation :

Consider following array

Iteration	Element Passed to Function	Value of Element
1	arr[0]	11
2	arr[1]	22
3	arr[2]	33
4	arr[3]	44
5	arr[4]	55

C Program to Pass Array to Function Element by Element :

```
#include<  stdio.h>
#include< conio.h>
void fun(int num)
{
printf("\nElement : %d",num);
 }
 void main()
{
int arr[5],i;
clrscr();
printf("\nEnter the array elements : ");
for(i=0;i< 5;i++)
scanf("%d",&arr[i]);
printf("\nPassing array element by element.....");
for(i=0;i< 5;i++)
fun(arr[i]);
getch();  }
```

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Output :

```
Enter the array elements : 1 2 3 4 5
Passing array element by
element.....  Element : 1
Element : 2
Element : 3
Element : 4
Element : 5
```

DISADVANTAGE OF THIS SCHEME :

- This type of scheme in which we are calling the function again and again but with **different array element is too much time consuming**. In this scheme we need to call function by pushing the current status into the system stack.

- It is better to pass complete array to the function so that we can save some system time required for pushing and popping.

18. RECURSION CONCEPT :

- Recursion is **basic concept** in Programming.
- When Function is defined in terms of itself then it is called as "**Recursion Function**".
- Recursive function is a **function which contains a call to itself**.

```
int factorial(int n)
{
if(n==0)
return(1);
else
return( n *
}
```

Factorial (n) = { 1 ; n * Factorial (n-1)

Example:

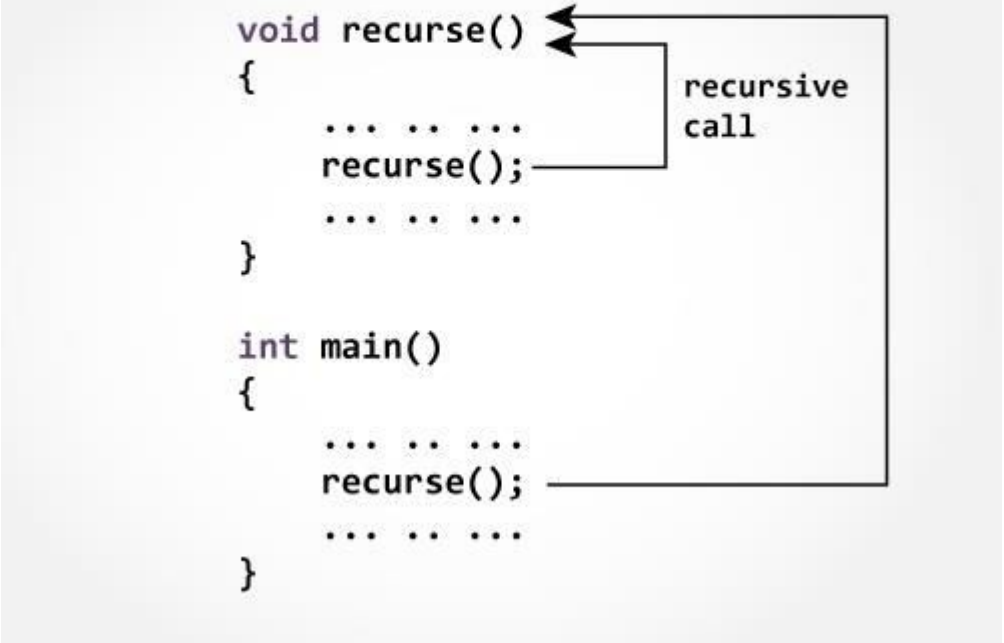
```
#include <stdio.h>
int sum(int n);
int main()
{
int number, result;
printf("Enter a positive integer: ");
scanf("%d", &number);
result = sum(number);
printf("sum=%d", result);
}
int sum(int num)
{
if (num!=0)
return num + sum(num-1); // sum() function calls itself
else
return num;
}
```

Output

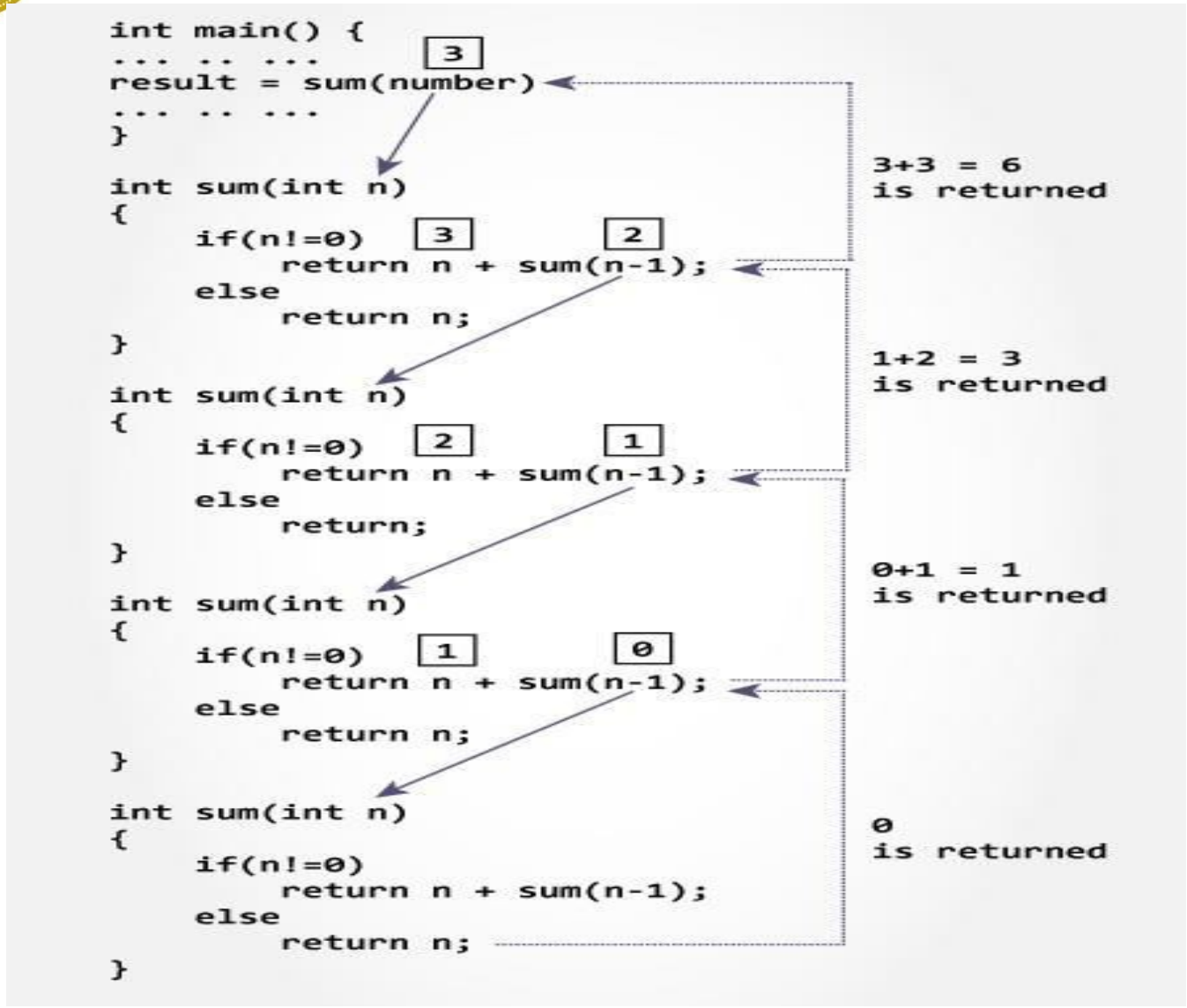
Enter a positive integer: 3 6

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

How does recursion work?



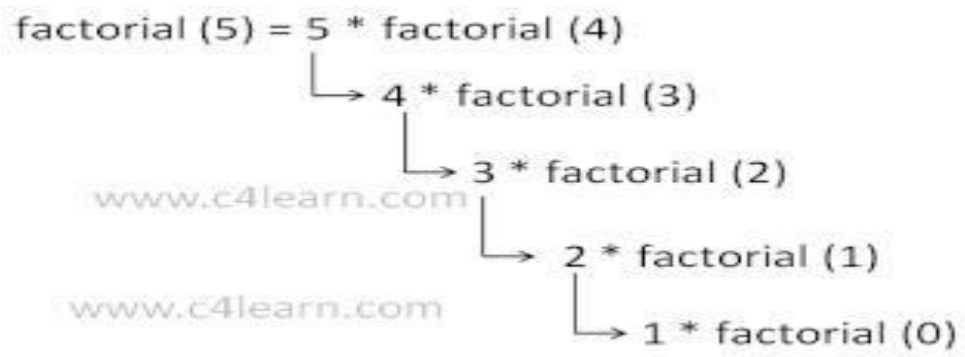
SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY,



SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Warning of using recursive function in C Programming :

- Recursive function must have at least one terminating condition that can be satisfied.
- Otherwise, the recursive function will call itself repeatably until the run time stack overflows.



SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY, CHENNAI.

Advantages and Disadvantages of Recursion

- Recursion makes program elegant and cleaner. All algorithms can be defined recursively which makes it easier to visualize and prove.
- If the speed of the program is vital then, you should avoid using recursion. Recursions use more memory and are generally slow. Instead, you can use loop.

Thank you