Principles of Programming using C(BPOPS103/203) SEM-1
VTU Question & Solution (2023-2024)
Prof. Rajeshwari R, Prof. Ann Mathews, Prof. Lynsha Helena Pratheeba HP
Department of CSE

MODULE-1

Q1 a.Define a computer. Explain the characteristics of a digital computer.

Ans:
Computer is a machine or electronic device that performs and manipulates information, and processes data on a program and also it have been a great source for communication.
Explanation: Characteristics: speed,accuracy,diligence,versatility,reliability,memory and automation.
1. Speed
Executing mathematical calculation, a computer works faster and more accurately than human. Computers have the ability to process so many millions (1,000,000) of instructions per second. Computer operations are performed in micro and nano seconds. A computer is a time saving device. It performs several calculations and tasks in few seconds that we take hours to solve. The speed of a computer is measure in terms of GigaHertz and MegaHertz.

2. Diligence
A human cannot work for several hours without resting, yet a computer never tires. A computer can conduct millions of calculations per second with complete precision without stopping. A computer can consistently and accurately do millions of jobs or calculations. There is no weariness or lack of concentration. Its memory ability also places it ahead of humans.

3. Reliability
A computer is reliable. The output results never differ unless the input varies. the output is totally depend on the input. when an input is the same the output will also be the same. A computer produces consistent results for similar sets of data, if we provide the same set of input at any time we will get the same result.

4. Automation
The world is quickly moving toward AI (Artificial Intelligence)-based technology. A computer may conduct tasks automatically after instructions are programmed. By executing jobs automatically, this computer feature replaces thousands of workers. Automation in computing is often achieved by the use of a program, a script, or batch processing.

5. Versatility
Versatility refers to a capacity of computer. Computer perform different types of tasks with the same accuracy and efficiency. A computer can perform multiple tasks at the same time this is known as versatility. For example, while listening to music, we may develop our project using PowerPoint and Wordpad, or we can design a website.

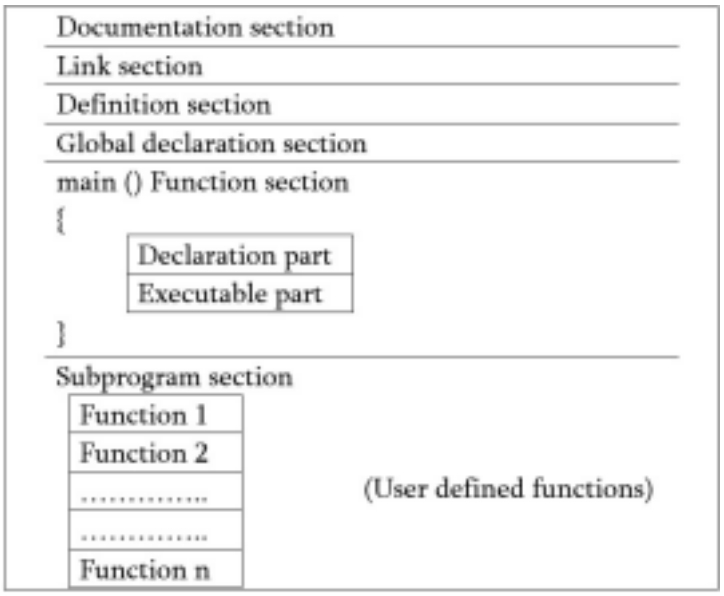
6. Memory

A computer can store millions of records. these records may be accessed with complete precision. Computer memory storage capacity is measured in Bytes, Kilobytes(KB), Megabytes(MB), Gigabytes(GB), and Terabytes(TB). A computer has built-in memory known as primary memory.

7. Accuracy
When a computer performs a computation or operation, the chances of errors occurring are low. Errors in a computer are caused by human's submitting incorrect data. A computer can do a variety of operations and calculations fast and accurately.

Q.1.b)Explain the basic structure of a C program with a neat diagram.

Ans:



Documentation Section:
This section is used to write Problem, file name, developer, date etc in comment lines within /*....*/ or separate line comments may start with // . Compiler ignores this section. Documentation enhances the readability of a program.
Link section :
To include header and library files whose in-built functions are to be used. Linker also required these files to build a program executable. Files are included with directive # Include Definition section: To define macros and symbolic constants by preprocessor directive #define.
Global section:
To declare global variables – to be accessed by all functions main() is the user defined function which is recognized by the compiler first. So, all C program must have user defined function main() { ..... }. It should have declaration part first then executable part.
Sub program section:
There may be other user defined functions to perform specific task when called.

/* Example: a program to find area of a circle – area.c - Documentation Section*/

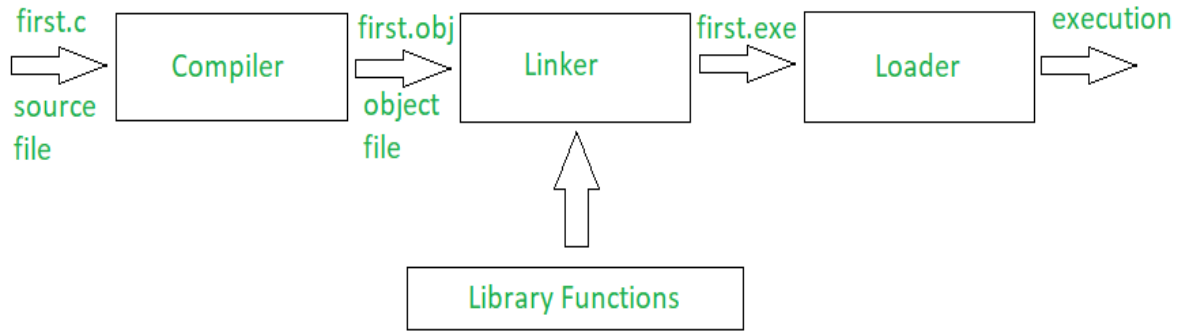
```
#include <stdio.h> /* - Link/Header Section */
#define PI 3.14 /* definition/global section*/
int main() /* main function section */
{
float r, area; /* declaration part */
printf("Enter radius of the circle : "); /* Execution part*/
scanf("%f", &r);
area=PI*r*r; /* using symbolic constant PI */
printf("Area of circle = %0.3f square unit\n", area);
return (0);
}
```

Q.2.a) With a neat diagram explain the steps in the execution of C program.

Ans:

A compiler converts a C program into an executable. There are four phases for a C program to become an executable:

- Pre-processing
- Compilation
- Assembly
- Linking



1. Pre-processing

This is the first phase through which source code is passed. This phase includes:

- Removal of Comments
- Expansion of Macros
- Expansion of the included files
- Conditional compilation

The preprocessed output is stored in the filename.I

2. Compiling

The next step is to compile filename.i and produce an; intermediate compiled output file filename.s. This file is in assembly-level instructions.

3. Assembling

In this phase the filename.s is taken as input and turned into filename.o by the assembler. This file contains machine-level instructions. At this phase, only existing code is converted into machine language.

4. Linking

This is the final phase is used to the creation of a single executable file from multiple object files.(.exe) which is ready to execute the file.

Q2 b. Explain the input and output statements in C with examples for each.
Ans:
In C, input involves providing data to a program, and output means displaying or sending data from the program. The stdio.h header file contains essential input and output functions, such as scanf() for reading data and printf() for displaying it.These all functions are collectively known as Standard I/O Library function.
Formatted I/O functions:
Formatted I/O functions which refers to an Input or Ouput data that has been arranged in a particular format. There are mainly two formatted I/O functions discussed as follows:

- printf()
- scanf()

printf():

printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).

Syntax 1:

To display any variable value.

```
printf("Format Specifier", var1, var2, …., varn);
```

Syntax 2:

To display any string or a message

```
printf("Enter the text which you want to display");
```

Example of printf()
printf("%d %c", info_a, info_b);

scanf():

scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that's why it is also a pre-defined function. In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

Syntax:

```
scanf("Format Specifier", &var1, &var2, …., &varn);
```

Example of scanf()
scanf("%d %c", &info_a,&info_b);

Example:

```
// C program to implement
// scanf() function
#include <stdio.h>
```

```
// Driver code
int main()
{
        int num1;

        // Printing a message on
        // the output screen
        printf("Enter a integer number: ");

        // Taking an integer value
        // from keyboard
        scanf("%d", &num1);

        // Displaying the entered value
        printf("You have entered %d", num1);

        return 0;
}
```

Output:
Enter a integer number: 56
You have entered 56

MODULE-2

Q.3.a) Explain the various operators in C.

Ans: give examples

ListofoperatorsusedinCLanguage:

i. Arithmetic Operators:
- + (addition)
- -(subtraction)
- * (multiplication)
- /(division)
- % (modulus)

ii. RelationalOperators:
- <(less than)
- <=(lessthanorequalto)
- >(greater than)
- >=(greaterthanor equalto)
- ==(equalto)
- !=(notequal to)

iii. Logical Operators:
- &&(and)
- || (or)
- ! (Not)

iv. Increment&Decrement:
- ++,--

v. Assignment Operator:
- =,+=, -=,*/,/=,%=

vi. Pre-processorOperator:
- #

vii. Bitwise Operators :

&(Bit-wiseAND)

|(Bit-wiseOR)

!(Bit-wiseNOT)

^(Bit-wiseXOR)

>>(Bit-wiserightshift)

<<(Bit-wiseleftshift)

viii. Ternary Operator(Conditionalif):

?:

ix. Addressof operator:

&

x. Pointer(dereference)operator:

*

xi. Comma Operator:

,

xii. Statement terminator operator:

;

Q3 b.Explain the different forms of if statements with flowchart.

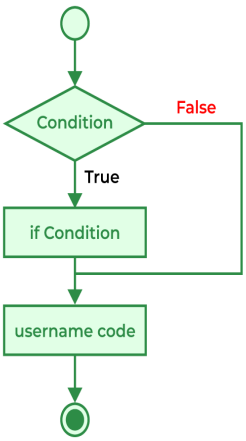
Ans:

1. if Statement
2. if-else Statement
3. Nested if Statement
4. if-else-if Ladder
5. switch Statement

1. if Statement:

The if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not.

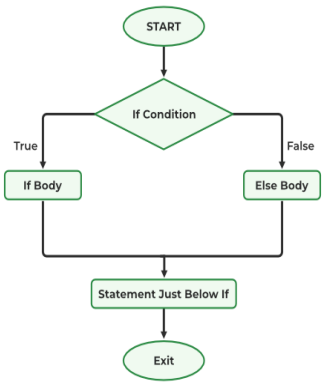
Flowchart:



2. if-else in C

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it execute false block.

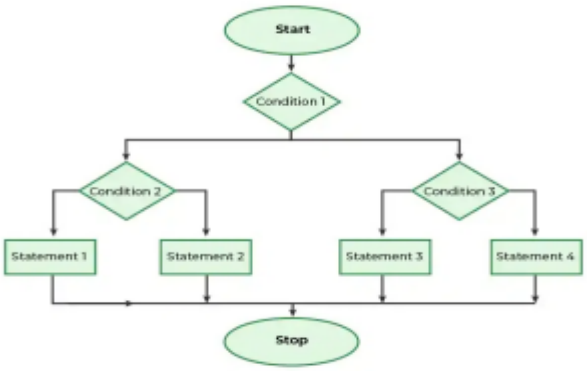
Flowchart:



3. Nested if-else in C

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, C allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

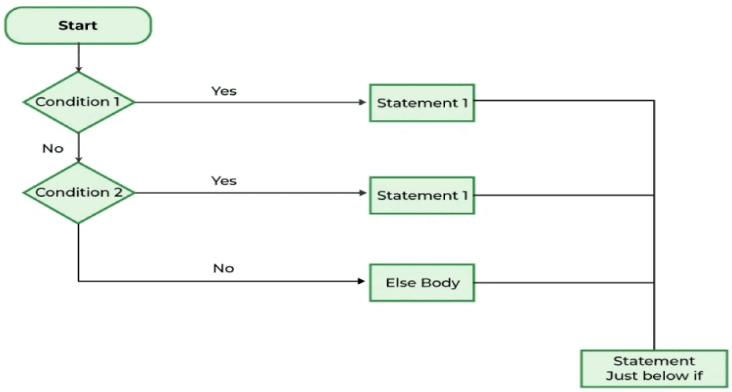
Flowchart:



4. if-else-if Ladder in C

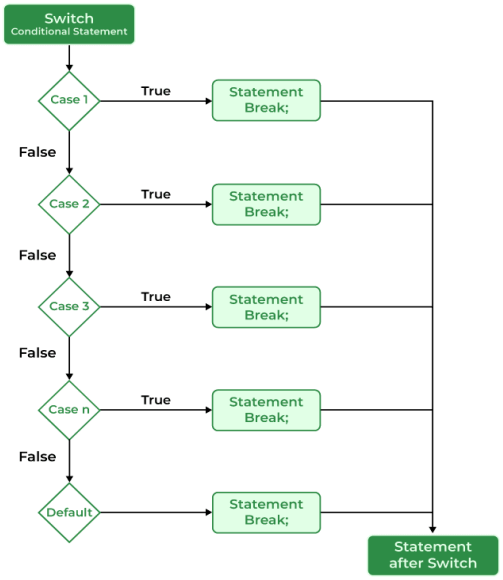
The if else if statements are used when the user has to decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. if-else-if ladder is similar to the switch statement.

Flowchart:



5. switch Statement in C

The switch case statement is an alternative to the if else if ladder that can be used to execute the conditional code based on the value of the variable specified in the switch statement. The switch block consists of cases to be executed based on the value of the switch variable.
Flowchart:



Q.4.a) Explain the switch statement with an example.
Ans:
switchstatement:
C has a built-in multiway decision statement known as a switch. It tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with the case is executed. The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring thecontroltothe nextstatementfollowingthe switch(where branchendswith}). The default is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values.
switchstatementsyntax:
switch(expression)
{ casecondition1
statement1;
statement2;
.....
break;
casecondition2
statement1;
statement2;
.....
break;
.....
default:
statement1;
statement2;
.....

```
}
/*Exampleofswitch*/
#include <stdio.h>
intmain()
{
intsalary,bonus;
char grade:
printf("Entergrade:");
scanf("%c", &grade);
printf("Entersalary:");
scanf("%d",&salary);
switch (grade)
{
case 'a':
case'A':bonus=salary;
break;
case'b':
case'B':bonus=salary+5000;
break;
default:bonus=salary+10000;/*lowergrade-morebonus*/
}
print("Bonus=%d\n",bonus);
return (0);
}
```

Q.4.b) Explain the break and continue statements with examples for each.

Ans:

breakstatementwouldonlyexitfromtheloopcontainingit.

```
//Exampleprogramtocheckanumberwhetheritisprime
#include<stdio.h>
intmain()
{
intn,d,prime=1;
printf("Enteranumber:");
scanf("%d",&n);
for(d=2;d<n/2;d++)
{
if(n%d==0)
prime=0;
break;/*thereisnoneedtocheckfurther*/
}
if(prime)
printf("Yes%disaprimenumber.\n",n);
else
printf("No,%disnotaprimenumber.\n",n);
return(0);
}
```

The continue statement is used in loops to skip the following statements in the loop and

tocontinue with the next iteration (current iteration in for loop).

```
//Exampleprogramtoentermarks(0-100)in6subjectsforsum #include<stdio.h>
intmain()
{
intmarks,sum=0,x;
for (x=1;x<=6;x++)
{
printf(“Entermarks%d:“,x)
scanf(“%d”,&marks);
if(marks<0||marks>100)
{
printf(“Invalidmarks!!!\n”);
continue;/*againreadmarksforsamepaper*/
}
sum+=marks;
}
printf(“sumofmarks=%d.\n”,sum);
return (0);
}
```

Q.4.c) Write a C program to find the largest three numbers using nested if statement.

Ans:

```
#include<stdio.h>
int main(){
   int a,b,c;
   printf("\nEnter Three Numbers : ");
   scanf("%d%d%d",&a,&b,&c);//100 98 105
   if(a>b){
      if(a>c){
         printf("\n%d is greatest number ",a);
      }else{
         printf("\n%d is greatest number ",c);
      }
   }else if(b>a){
      if(b>c){
         printf("\n%d is greatest number ",b);
      }else{
         printf("\n%d is greatest number ",c);
      }
   }else{
      printf("\n%d is greatest number ",a);
   }
   return 0;
}
```

Output:Enter Three Numbers : 25
36
20
36 is greatest number

MODULE-3

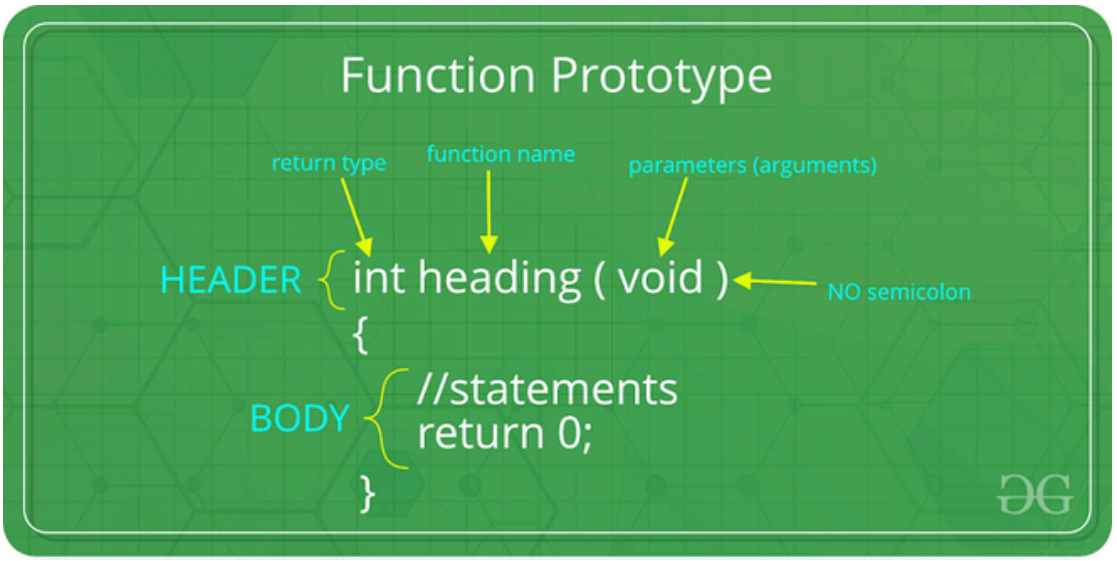
Q.5.a)Discuss in detail the parts of user defined function?

A **user-defined function** is a type of function in C language that is defined by the user himself to perform some specific task. It provides code reusability and modularity to our program. User-defined functions are different from built-in functions as their working is specified by the user and no header file is required for their usage.
The user-defined function in C can be divided into three parts:

1. Function Prototype
2. Function Definition
3. Function Call

Function Prototype
A function prototype is also known as a function declaration which specifies the **function's name, function parameters,** and **return type**. The function prototype does not contain the body of the function. It is basically used to inform the compiler about the existence of the user-defined function which can be used in the later part of the program.

return_type **function_name** (type1 *arg1*, type2 *arg2*, ... typeN *argN*);



Function Definition
Once the function has been called, the function definition contains the actual statements that will be executed. All the statements of the function definition are enclosed within **{ } braces.**

Syntax

return_type **function_name** (type1 *arg1*, type2 *arg2* .... typeN *argN*) {

```
    // actual statements to be executed
    // return value if any
}
```

C Function Call

In order to transfer control to a user-defined function, we need to call it. Functions are called using their names followed by round brackets. Their arguments are passed inside the brackets.

Syntax

function_name(arg1, arg2, ... argN);

Example of User-Defined Function

```
// C Program to illustrate the use of user-defined function
#include <stdio.h>
// Function prototype
int sum(int, int);
// Function definition
int sum(int x, int y)
{
    int sum;
    sum = x + y;
    return x + y;
}
 // Driver code
int main()
{
    int x = 10, y = 11;
    // Function call
    int result = sum(x, y);
    printf("Sum of %d and %d = %d ", x, y, result);
    return 0;
}
```

Output

Sum of 10 and 11 = 21

Q.5.b) Discuss the storage classes in C.

Ans:

In C programming, storage classes are used to define the scope, lifetime, and initial value of variables. There are four storage classes in C:

Automatic Storage Class (auto):

- Variables declared within a function without using any storage class specifier are considered automatic by default.

- They are created when the function is called and destroyed when the function exits.
- Their initial values are garbage unless initialized explicitly.

Example:

```
#include<stdio.h>

void function() {
   auto int x = 10; // automatic variable
   printf("Value of x inside function: %d\n", x);
}

int main() {
   function();
   // printf("Value of x outside function: %d\n", x); // This will result in an error as x is not accessible here
   return 0;
}
```

Static Storage Class (static):

- Variables declared with the static keyword inside a function retain their values between function calls.
- Static variables are initialized only once, before the program starts execution.
- They exist throughout the lifetime of the program but are only accessible within the function where they are declared.
- When declared at the global level, static variables are accessible only within the file they are defined in.

Example:

```
#include<stdio.h>

void function() {
   static int x = 10; // static variable
   printf("Value of x inside function: %d\n", x);
   x++; // Value persists across function calls
}

int main() {
   function(); // Output: Value of x inside function: 10
   function(); // Output: Value of x inside function: 11
   return 0;
}
```

Extern Storage Class (extern):

- Variables declared with the extern keyword are not allocated any storage space.
- They are typically used for declaring variables in one file that are defined in another file.

- The actual variable is declared and defined elsewhere, typically in another source file or library.
- Extern variables must be declared before they are used in the file.

Example:

```
// File: file1.c
#include<stdio.h>

int x = 10; // global variable

// File: file2.c
#include<stdio.h>

extern int x; // Declaration of the global variable defined in file1.c

int main() {
   printf("Value of x: %d\n", x); // Output: Value of x: 10
   return 0;
}
```

Register Storage Class (register):

- The register keyword is used to suggest that a variable should be stored in a register of the CPU for faster access.
- It's a hint to the compiler for optimization purposes, but it's not guaranteed that the variable will be stored in a register.
- The address of a register variable cannot be accessed.

These storage classes provide programmers with flexibility in managing memory and controlling variable behavior within a program. Each class has its specific use cases and implications on variable lifetime and scope.

Example:

```
#include<stdio.h>

int main() {
   register int x = 10; // register variable
   printf("Value of x: %d\n", x);
   return 0;
}
```

Q.6.a)Define recursion?Write a c program to Find the factorial of n using recursion?

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

eg:

```
#include<stdio.h>
#include<math.h>
int sum(int k);
int main() {
```

```
  int result = sum(10);
  printf("%d", result);
  return 0;
}

int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }
}
```

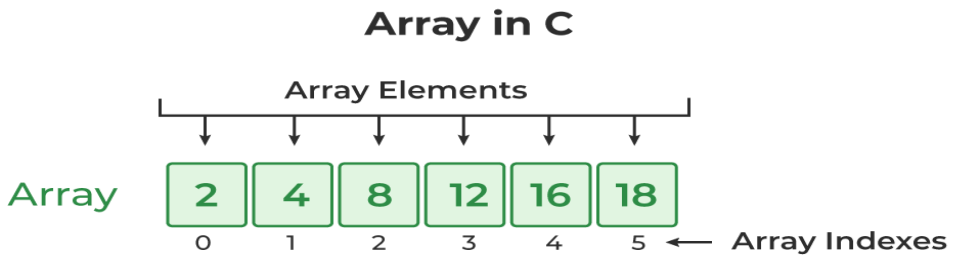
Factorial of N with recursion

```
#include<stdio.h>

long factorial(int n)

{

 if (n == 0)

   return 1;

 else

   return(n * factorial(n-1));

}

void main()

{

 int number;

 long fact;

 printf("Enter a number: ");

 scanf("%d", &number);

  fact = factorial(number);

 printf("Factorial of %d is %ld\n", number, fact);

 return 0;

}
```

Q.6.b) What is an array? Explain the declaration and initialization of 1-D array?

Array in C is one of the most used data structures in C programming. It is a simple and fast way of storing multiple values under a single name.

An array is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.



Declaration of 1-D array

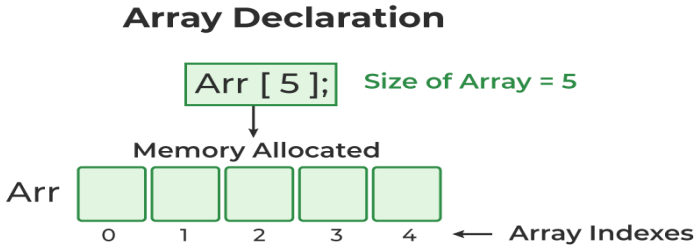
Syntax of Array Declaration

data_type array_name [size];

or

data_type array_name [*size1*] [*size2*]...[*sizeN*];

where N is the number of dimensions.



Example of Array Declaration

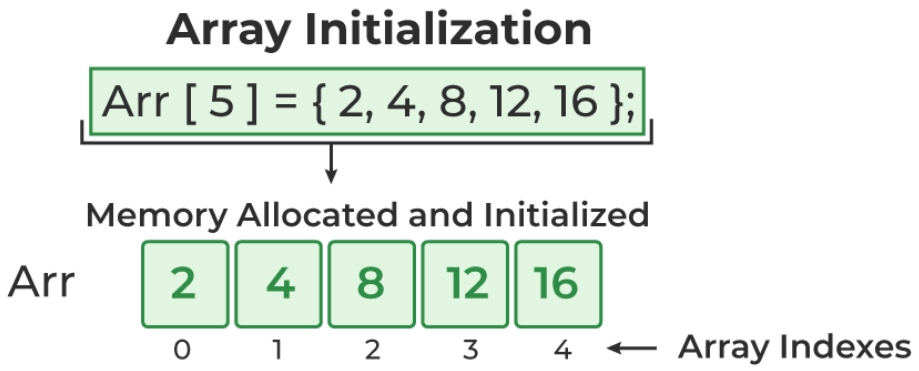
```
#include <stdio.h>

int main()
{

   // declaring array of integers
   int arr_int[5];
   // declaring array of characters
```

```
    char arr_char[5];

    return 0;
}
```

Array Initialization

An initializer list is the list of values enclosed within braces { } separated b a comma.

data_type array_name [*size*] = {value1, value2, ... valueN};



Array Initialization with Declaration without Size

data_type array_name[] = {1,2,3,4,5};

Array Initialization after Declaration (Using Loops)

We initialize the array after the declaration by assigning the initial value to each element individually. We can use for loop, while loop, or do-while loop to assign the value to each element of the array.

```
for (int i = 0; i < N; i++) {
    array_name[i] = valuei;
}
```

Example of Array Initialization in C

```
#include <stdio.h>

int main()
{
```

```
    // array initialization using initialier list
    int arr[5] = { 10, 20, 30, 40, 50 };

    // array initialization using initializer list without
    // specifying size
    int arr1[] = { 1, 2, 3, 4, 5 };

    // array initialization using for loop
    float arr2[5];
    for (int i = 0; i < 5; i++) {
        arr2[i] = (float)i * 2.1;
    }
    return 0;
}
```

Q.6.c) Write a C program to perform matrix multiplication?

Ans:

```
#include<stdio.h>

#include<stdlib.h>

int main(){

int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;

system("cls");

printf("enter the number of row=");

scanf("%d",&r);

printf("enter the number of column=");

scanf("%d",&c);

printf("enter the first matrix element=\n");

for(i=0;i<r;i++)

{

for(j=0;j<c;j++)

{

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

```
}
}
printf("enter the second matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
//for printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
```

```
printf("%d\t",mul[i][j]);

}

printf("\n");

}

return 0;

}
```

Output:

```
enter the number of row=3
enter the number of column=3
enter the first matrix element=
1 1 1
2 2 2
3 3 3
enter the second matrix element=
1 1 1
2 2 2
3 3 3
multiply of the matrix=
6 6 6
12 12 12
18 18 18
```

MODULE - 4

Q.7.a) Write functions to implement string operations such as compare concatenate and string length. Convince the parameter passing techniques.

Ans:

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

// Function to compare two strings
int stringCompare(const char *str1, const char *str2) {
   return strcmp(str1, str2);
}

// Function to concatenate two strings
void stringConcatenate(const char *str1, const char *str2, char *result) {
   strcpy(result, str1);
   strcat(result, str2);
}

// Function to calculate the length of a string
```

```
int stringLength(const char *str) {
   return strlen(str);
}

int main() {
   char str1[100] = "Hello";
   char str2[100] = "World";
   char result[200];

   // Comparing strings
   int comparison = stringCompare(str1, str2);
   if (comparison == 0)
      printf("Strings are equal\n");
   else if (comparison < 0)
      printf("String 1 is less than String 2\n");
   else
      printf("String 1 is greater than String 2\n");

   // Concatenating strings
   stringConcatenate(str1, str2, result);
   printf("Concatenated string: %s\n", result);

   // Calculating string lengths
   printf("Length of String 1: %d\n", stringLength(str1));
   printf("Length of String 2: %d\n", stringLength(str2));

   return 0;
}
```

Output:
String 1 is less than String 2
Concatenated string: HelloWorld
Length of String 1: 5
Length of String 2: 5

Q.7.b) Develop a program using pointers to compute, sum, mean and standard devíation of all the elements stored in an array.

Ans:

```
#include<stdio.h>
#include<math.h>
int main()
{
        int i,n;
        float a[10],mean,sd,sum,var;
        float *p;                   // p is a pointer to float value

        printf("\n Enter Number of elements :");
        scanf("%d",&n);
        printf("\n Enter the elements :");
```

```
        p=a;                    // pointer p points to first element of a
        for(i=0;i<n;i++)
        {
                scanf("%f",p);
                p++;            // pointer p points to the next element of the array
        }

        p=a;                    // Initialize p to the first element of the array
        printf("\n input Elements are:\n");
        for(i=0;i<n;i++)
        {
                printf("%f",*p);
                p++;            // Pointer p is made to point to the next element
        }

        p=a;                    // Initialize p to the first element of the array

        sum=sd=mean=var=0;

        // Find the sum of the array elements
        for(i=0;i<n;i++)
        {
                sum=sum+(*p);
                p++;
        }
        // Find the mean
        mean=sum/n;

        // Find variance
        p=a;
        for(i=0;i<n;i++)
        {
                var=var+pow((*p-mean),2);
                p++;
        }
        var=var/n;

        // Find Standard Deviation
        sd=sqrt(var);

        // Print Sum, mean and Standard Deviation
        printf("\n\n mean=%f\nsum=%f\nsd=%f\nvar=%f\n",mean,sum,sd,var);
        return 0;
}
```

Output:

```
Enter Number of elements : 5

Enter the elements : 1 2 3 4 5

Input Elements are:
```

1.0000002.0000003.0000004.0000005.000000

Mean=3.000000
Sum=15.000000
Standard Deviation=1.414214
Variance=2.000000

Q.8.a) Define a pointer, Discuss the declaration of pointer variables.

Ans:

A pointer in C is a variable that stores the memory address of another variable. Pointers are widely used in C for various purposes such as dynamic memory allocation, passing addresses to functions, and accessing data structures like arrays and linked lists.

Declaration of Pointer Variables:
The declaration of a pointer variable in C consists of specifying the data type of the variable it points to, followed by an asterisk (*) and the name of the pointer variable.
Syntax:
datatype *pointer_name;

Here, datatype specifies the type of data that the pointer will point to. * indicates that the variable is a pointer. pointer_name is the name of the pointer variable.
Example:

```
#include<stdio.h>
int main() {
   int num = 10; // declare and initialize an integer variable
   int *ptr;     // declaration of a pointer variable

   ptr = &num;   // assigning the address of 'num' to the pointer variable

   printf("Value of num: %d\n", num);   // Output: Value of num: 10
   printf("Address of num: %p\n", &num); // Output: Address of num: <memory
address>

   printf("Value of ptr: %p\n", ptr);   // Output: Value of ptr: <memory address of
num>
   printf("Value pointed by ptr: %d\n", *ptr); // Output: Value pointed by ptr: 10

   return 0;
}
```

In this example:

- We declare an integer variable num and initialize it to 10.
- Then, we declare a pointer variable ptr using the syntax int *ptr;. This pointer will store the address of an integer variable.
- We assign the address of num to the pointer variable ptr using the address-of operator &.

- We print the value of num, its address, the value of ptr, and the value pointed by ptr.

Q.8.b) Discuss the various string handling functions in C.

Ans:

In C programming, strings are handled using an array of characters terminated by a null character ('\0'). There are several standard library functions provided by C for string handling operations. Here are some commonly used string handling functions along with examples:

strlen() - Calculate Length of String:

- size_t strlen(const char *str);
- Calculates the length of the string str excluding the null terminator.

Example:

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

int main() {
   char str[] = "Hello, world!";
   size_t len = strlen(str);
   printf("Length of '%s' is: %zu\n", str, len);
   return 0;
}
```

Output:
Length of 'Hello, world!' is: 13

strcpy() - Copy Strings:

- char *strcpy(char *dest, const char *src);
- Copies the string src to dest including the null terminator.

Example:

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

int main() {
   char src[] = "Hello";
   char dest[10];
   strcpy(dest, src);
   printf("Copied string: %s\n", dest);
   return 0;
}
```

Output:
Copied string: Hello

strcmp() - Compare Strings:

- int strcmp(const char *str1, const char *str2);
- Compares the strings str1 and str2 lexicographically.

Example:

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

int main() {
   char str1[] = "apple";
   char str2[] = "banana";
   int result = strcmp(str1, str2);
   if (result < 0)
      printf("%s comes before %s\n", str1, str2);
   else if (result > 0)
      printf("%s comes after %s\n", str1, str2);
   else
      printf("%s and %s are equal\n", str1, str2);
   return 0;
}
```

Output:
apple comes before banana

strcat() - Concatenate Strings:

- char *strcat(char *dest, const char *src);
- Concatenates the string src to the end of dest.

Example:

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

int main() {
   char dest[20] = "Hello";
   char src[] = ", world!";
   strcat(dest, src);
   printf("Concatenated string: %s\n", dest);
   return 0;
}
```

Output:
Concatenated string: Hello, world!

Q.8.c) Write a C program to swap two Numbers using call by reference technique.

Ans:

```
#include <stdio.h>
// Function to swap two numbers using call by reference
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
```

```
}
int main() {
   int num1, num2;
   // Input two numbers from the user
   printf("Enter two numbers: ");
   scanf("%d %d", &num1, &num2);

   // Display the numbers before swapping
   printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);

   // Call the swap function passing the addresses of the numbers
   swap(&num1, &num2);

   // Display the numbers after swapping
   printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);

   return 0;
}
```

Output:
Enter two numbers: 10 20
Before swapping: num1 = 10, num2 = 20
After swapping: num1 = 20, num2 = 10

MODULE - 5

Q.9.a) Define a structure. Explain the types of structure declaration with examples for each.

Ans:
Structure is a data structure whose individual elements can differ in type. It may contain integer
elements, character elements, pointers, arrays and even other structures can also be included as
elements within a structure. struct is keyword to define a structure.
structtag{typemember1;
typemember2;
typemember3;typemembern;};
New Structure Type Variable Scan Be Declared As Follows:
struct tag var1, var2, var3,.....varn,var[10];
i) Array Of Structure:
Whole structure can be an element of the array. A student structure with members roll, name and marks:

```
structstudent
{
introll;
charname[30];
int marks;
};
```

Now we can use this template to create array of 60 students for their individual roll,name and marks.
structstudents[60];/*array of structure*
toaccessrollofstudentx:s[x].roll
ii) Array Within Structure:
A member of structure may be array. In above example name is also array of characters. We can create a structure of student with members roll, name & marks (array of integers) for 6 papers:

```
structstudent
{
introll;
charname[30];
intmarks[6];/*array within structure*/
}s[60];
```

Toaccessthemarksofstudentxinpapery:s[x].marks[y]
ii)StructurewithinStructure:
A structure may be defined within another structure or a structure variable may be declared as a member within another structure.Here datestructure is defined within employee structure:

```
structemployee
{
int eno;
charname[30];
longint salary;
structdate/*structure within structure*/
{
int dd,mm, yy;
}dob,doj;
}e[1000];
```

Orstructurevariableofdatestructurecanbedefinedwithinemployeeifdatestructureis separately
defined:

```
structdate
{
int dd,mm, yy;
};
structemployee
{
int eno;
charname[30];
longint salary;
struct date dob,doj;/*variable of data structure within employee*/
}e[1000];
```

To access day(dd) of date structure we will use member operator (.) dot operator in following way:
e[x].dob.dd

Q.9.b) Implement structures to read, write and compute average marks and the

students scoring below and above average in a class of 'N students.

Ans:

```
#include<stdio.h>
 struct student
        {
                int id;
                char name[20];
                float sub[6];
                float avg;
        };
 int main()
{
        struct student s[20];
        float sum=0;
        int i,j,n;

        // Accept the number of records/students
        printf("Enter the number of records :");
        scanf("%d",&n);

        // Accept data for all the fields/members of each record
        printf("Enter %d student details...\n",n);

        for(i=0;i<n;i++)
        {
                printf("\n\nEnter student ID, name :");          // Student ID
                scanf("%d%s",&s[i].id, s[i].name);

                printf("Enter 6 subject marks :");

                for (j=0;j<6;j++)
                        {
                                scanf("%f", &s[i].sub[j]);
                        }
        }

        // Compute the average of each student

        for(i=0;i<n;i++)
        {
                sum=0;
                for (j=0;j<6;j++)
                        {
                                sum = sum + s[i].sub[j];
                        }
                s[i].avg = sum / 6;
        }

        // Display student ID, name and average of all students
        // who have scored above average marks
```

```
        printf("Students scoring above the average marks....\n");
        printf("\n\nID\tName\tAverage\n\n");

        for(i=0;i<n;i++)
        {
                if(s[i].avg>=35.0)
                printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
        }

        // Display student ID, name and average of all students
        // who have scored below average marks

        printf("\n\nStudents scoring below the average marks....\n");
        printf("\n\nID\tName\tAverage\n\n");

        for(i=0;i<n;i++)
        {
                if(s[i].avg<35.0)
                printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
        }

        return 0;
}
```

Output:

```
Enter the number of records : 3
Enter 3 student details...
Enter student ID, name : 101 John
Enter 6 subject marks : 78 85 92 80 88 90
Enter student ID, name : 102 Alice
Enter 6 subject marks : 85 76 80 88 72 81
Enter student ID, name : 103 Bob
Enter 6 subject marks : 70 65 68 72 62 74
Students scoring above the average marks....
ID   Name    Average
101  John    85.500000
102  Alice   80.333336
Students scoring below the average marks....
ID   Name    Average
103  Bob     68.500000
```

Q.10.a) Differentiate between structures and union.

Ans:

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Q.10.b) Define a structure by name DOB consisting of three members dd, mm and yy, Develop a C program that would read values to the individual member and display the date in the form dd/mm/yyyy.

Ans:

```
#include <stdio.h>
// Define structure DOB
struct DOB {
    int dd; // day
    int mm; // month
    int yy; // year
};
int main() {
    // Declare a variable of type DOB
    struct DOB date;

    // Input values for day, month, and year
    printf("Enter day, month, and year (in dd mm yyyy format): ");
    scanf("%d %d %d", &date.dd, &date.mm, &date.yy);

    // Display the date in dd/mm/yyyy format
    printf("Date: %02d/%02d/%04d\n", date.dd, date.mm, date.yy);

    return 0;
}
```

Output:

```
Enter day, month, and year (in dd mm yyyy format): 25 12 2023
Date: 25/12/2023
```

Q.10.c) Explain the various file operations with syntax for each.

Ans:
In C programming, file operations are performed using the standard I/O library functions defined in <stdio.h>. These functions allow you to perform various operations on files such as opening, closing, reading, and writing. Here are the commonly used file operations along with their syntax:

Opening a File:

- To open a file, you use the fopen() function.
- Syntax: FILE *fopen(const char *filename, const char *mode);
- Example: FILE *file = fopen("example.txt", "r");
- Here, "example.txt" is the filename, and "r" is the mode indicating read access.

Closing a File:

- To close a file, you use the fclose() function.
- Syntax: int fclose(FILE *stream);
- Example: fclose(file);
- Here, file is the file pointer returned by fopen().

Reading from a File:

- To read from a file, you use the fscanf() or fgets() function.
- Syntax (fscanf): int fscanf(FILE *stream, const char *format, ...);
- Syntax (fgets): char *fgets(char *str, int n, FILE *stream);
- Example (fscanf): fscanf(file, "%d %s", &num, str);
- Example (fgets): fgets(buffer, sizeof(buffer), file);

Writing to a File:

- To write to a file, you use the fprintf() or fputs() function.
- Syntax (fprintf): int fprintf(FILE *stream, const char *format, ...);
- Syntax (fputs): int fputs(const char *str, FILE *stream);
- Example (fprintf): fprintf(file, "%d %s\n", num, str);
- Example (fputs): fputs("Hello, world!\n", file);

Checking for End-of-File (EOF):

- To check for the end-of-file (EOF) condition, you use the feof() function.
- Syntax: int feof(FILE *stream);
- Example: if (feof(file)) { /* End of file reached */ }

Error Handling:

- To check for errors while performing file operations, you use the ferror() function.
- Syntax: int ferror(FILE *stream);
- Example: if (ferror(file)) { /* Error occurred */ }

These are some of the commonly used file operations in C programming. They allow you to perform various tasks such as reading data from files, writing data to files, and handling errors during file operations.