Model Answer for M.Sc. First Semester,
Subject: Programming in C Language
SECTION –A

Q. No. 1. Attempt the following (Consider header file included):

(I) Write the use of const modifier.

Ans:

Const Modifier is used to ensure that your program does not inadvertently alter a variable that you intended to be a constant. It also reminds anybody reading the program listing that the variable is not intended to change. We can use **const** in several situations.

Ex.

```
const int PI=3.14;
int * const ptr=3.14;
const int * ptr=3.14;
const int * const ptr=3.14;
```

(II) Write syntax of do-while loop.

Ans:

The Do Statement: It is an exit-controlled loop statement. In **while** statement, the body of loop does not executed if the test-expression evaluate to false at the very first attempt. Sometimes, it might be necessary to execute the body of the loop before the test is performed. **Do-while** statement used in this situation.

Syntax:

```
do
{
  body of the loop
}
while(test-expression);
```

First the blocks of statements are executed at the end of loop while statement is executed. If the test-expression evaluated to true (non-zero) then program control goes to evaluate the body of a loop once again. This process continues till test-expression evaluates to true. When it becomes false, then the loop terminates.

Note: The while statement should be terminated with ";" (semicolon).

(III) Write the various types of arguments.

Ans:

The mechanism used to convey information to the function is the 'argument'. There are two types of arguments.

Formal argument: the argument used at the time of function definition
Ex.

```
 void display( char * message)
{
        ……
        Function body
        ……
}
```

Message is a formal argument in this function

Actual argument: the argument passed at the time of function call

```
#include<stdio.h>
void main()
{
        char msg[10]= {'H','e','l','l',o'};
        display(msg);// msg is actual argument
}
```

(IV) What are various ways to include files?

Ans:

There are two ways to include files:

#include "filename": this command would look for the file **goto.c** in the current directory as well as the specified list of directories as mentioned in the include search path that might have been set up.

#include <filename>: this command would look for the file **goto.c** in the specified list of directories only.

(V) Explain fseek() and ftell() function.

Ans:

fseek(): sets the file position of the stream to the given offset. The argument *offset* signifies the number of bytes to seek from the given *whence* position.

Function prototype/syntax: int fseek(FILE *stream, long int offset, int whence)

ftell(): returns the current file position of the given stream.

Function prototype/syntax: long int ftell(FILE *stream)

(VI). What will be the output of the following program?

```
void main()
{
        int x=5,y=0,z=10;
        if(++x>5 || y++<z)
        printf("TRUE");
        printf(" x=%d, y=%d, z=%d",x,y,z);
}
```

Output: TRUEx=6, y=0, z=10

(VII). What will be the output of the following program?

```
void main( )
{
        int i, a[ ] = { 2, 4, 6, 8, 10 } ;
        for ( i = 0 ; i <= 4 ; i++ )
        {
                i[a]=a[i]+i++;
                printf( "%d, %d\n", a[i],*(a+i) ) ;
        }
 }
```

Output:

4, 4

8, 8

0, 0

(VIII). The macro FILE is defined in which of the following files:

1. stdlib.h

2. stdio.c

3. io.h

4. stdio.h

Ans: (4) stdio.h

(IX). Point out the errors, if any, in the following programs:

```
main( )
{
        struct employee
        {
                char name[25] ;
                int age ;
                float bs ;
        } ;
        struct employee e ;
        strcpy ( e.name, "Hacker" ) ;
        age = 25 ;
        printf ( "\n%s %d", e.name, age ) ;
}
```

Ans: Error: Undeclared age

(VII). What will be the output of the following program?

```
main( )
{
        int i = 5, j = 2 ;
        junk ( &i, &j ) ;
        printf ( "\n%d %d", i, j ) ;
}
junk ( int *i, int *j )
{
        *i = *i * *i ;
        *j = *j * *j ;
}
```

Output: 25, 4

SECTION-B

Q. No. 2. Write a program to compute and display the sum of all integers that are divisible by 6 but not divisible by 4 and lie between 0 and 100. The program should also count and display the number of such value.

Ans:

```
#include<stdio.h>
#include<conio.h>
main()
{
        int i,sum=0,count=0;
        for(i=1;i<=100;i++)
        {
                if(i%6==0 && i%4!=0)
                {
                        printf("\n %d",i);
                        sum+=i;
                        count++;
                }
        }
        printf("\nSum is %d",sum);
        printf("\ncount is %d",count);
        getch();
        return 0;
}
```

Q. No. 3. What are built-in functions in C Language? Describe any five built-in Input functions with suitable example.

Ans:

Function: Function is a self contained block of statement to perform a specific task.

There are two type of function in C programming Language

- Built-in or Standard library function
- User defined function

Built-in or Library function: function provided in the standard library of the C compiler. Prototype of these function cannot be modified they can be called/used by including the respective header file in which there are declared and defined.

Five built in function:

There are numerous library functions available for Input. These can be classified into three broad categories:

(a) Console Input functions- Functions to receive input from keyboard and write output to VDU.
(b) File Input functions- Functions to perform input operations on a floppy disk or hard disk.

(1) Scanf( ): Reads formatted input from stdin file stream allows us to enter data from keyboard that will be formatted in a certain way.
Function prototype: int scanf(const char *format, ...)
In other words, general form of **scanf( )** statement is as follows: scanf ("format string", list of addresses of variables) ;

Ex.
scanf ("%d %f %c", &c, &a, &ch) ;

(2) getch(): Reads a character directly from the console without buffer, and without echo.

Function prototype: int getchar(void)

(3) getche(): Reads a character directly from the console without buffer, but with echo.

Function prototype: int getchar(void)

(4) getche(): Gets a character (an unsigned char) from stdin.

Function prototype: int getchar(void)

Ex.

The following example shows the usage of getch(),getche() and getchar() function.

```
main( )
{
        char ch ;
        printf ( "\nPress any key to continue" ) ;
        getch( ) ; /* will not echo the character */
        printf ( "\nType any character" ) ;
        ch = getche( ) ; /* will echo the character typed */
        printf ( "\nType any character" ) ;
        getchar( ) ; /* will echo character, must be followed by enter key */
        printf ( "\nContinue Y/N" ) ;
        fgetchar( ) ; /* will echo character, must be followed by enter key */

}
```

(5) gets (): reads a line from stdin and stores it into the string pointed to by str. It stops when either the
newline character is read or when the end-of-file is reached, whichever comes first.

Function prototype: char *gets(char *str)

Ex.

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

```
int main()
{
        char str[50];

        printf("Enter a string : ");
        gets(str);
        printf("You entered: %s", str);
        return(0);
}
```

(6) getc (): gets the next character (an unsigned char) from the specified stream and advances the position indicator for the stream.

Function prototype: int getc(FILE *stream)

Ex.

```
#include<stdio.h>
int main()
{
        char c;
        printf("Enter character: ");
        c = getc(stdin);
        printf("Character entered: ");
        putc(c, stdout);
        return(0);
}
```

Q. No. 4. Define a structure data type called **time_struct** containing three members **integer** hour, **integer** minute and **integer** second. Develop a program that would assign values to the individual members and display the time in the following form: 16:40:51(hh:mi:ss) as well as make functions to input values to the members and to display time in the given format.

Ans:

```
        #include<stdio.h>
        #include<conio.h>
        struct time_struct
        {
                int hour;
                int minute;
                int second;
        };
        struct time_struct readtime()
        {
             int vldflag;
             struct time_struct tmptime;
             vldflag=scanf("%2d:%2d:%2d",&tmptime.hour,&tmptime.minute,
        &tmptime.second);
             if(vldflag!=3)
             {
                 printf("Invalid time\n") ;
             } return tmptime;
```

```
}
void displaytime(struct time_struct tmptime)
{
        printf("Time is %2d:%2d:%2d",tmptime.hour,tmptime.minute,tmptime.second);
}
main()
{
    struct time_struct mytime;
    mytime=readtime();
    displaytime(mytime);
    getch();
    return 0;
}
```

Q. No. 5. Write short notes on the following:

(a) Void Pointer (b) Pointer to Pointer (c) free() function (d) realloc() function (e) Pointer Comparison

Ans:

(a) Void Pointer: In C pointers are variables that store addresses and can be *null*. Each pointer has a type it points to, but one can freely cast between pointer types (but not between a function pointer and non-function pointer type). A special pointer type called the "void pointer" allows pointing to any (non-function) variable type, but is limited by the fact that it cannot be **dereferenced** directly.

void pointer or **void*** : is supported in ANSI C as a generic pointer type. A pointer to void can store an address to any non-function data type, and, in C, is implicitly converted to any other pointer type on assignment, but it must be explicitly cast if **dereferenced** inline.

Ex.

```
void main()
{
        int *intptr, x = 4,;
        void* voidptr;
        voidptr=&x;
        printf("%d",*voidptr); //invalid
        intptr=voidptr;  /* void* implicitly converted to int*: valid C, but not C++ */
        printf("%d",*intptr); //valid
        voidptr=intptr; //valid
        printf("%d",*voidptr); // again invalid
}
```

(b) Pointer to Pointer: when a pointer variable store the address of other pointer variable.

Ex.

```
void main()
{
        int *intptr, **intptr_to_ptr, x = 4,;
```

```
        intptr=&x;
        intptr_to_ptr=&intptr;
        printf("%d",*intptr); //  display value of  x
        printf("%d",**intptr_to_ptr); //  display value of
        printf("%d",*intptr); //display address of x variable
        printf("%d",*intptr_to_ptr); //display address of intptr pointer variable
}
```

(c) free() function: de-allocates the memory previously allocated by a call to calloc(), malloc() or realloc().

Declaration

void free(void *ptr)

Parameters

ptr -- T his is the pointer to a memory block previously allocated with malloc(), calloc() or realloc() to be de-allocated. If a null pointer is passed as argument, no action occurs.

Return Value

This function does not return any value.

(d) realloc() function: attempts to resize the memory block pointed to by ptr that was previously allocated with a call to malloc() or calloc(). If ptr is NULL, realloc() is identical to malloc()

Declaration

void *realloc(void *ptr, size_t size)

Parameters

ptr -- This is the pointer to a memory block previously allocated with malloc, calloc or realloc to be reallocated. If this is NULL, a new block is allocated and a pointer to it is returned by the function.

size -- This is the new size for the memory block, in bytes.If it is 0 and ptr points to an existing block of memory, the memory block pointed by ptr is de-allocated and a NULL pointer is returned.

Return Value

This function returns a pointer to the newly allocated memory, or NULL if the request fails.

Ex. showing usage of free() and realloc() but not compulsory

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        char *str;
        str = (char *) malloc(15);
        strcpy(str, "gurughasidas");
        printf("String = %s, Address = %u\n", str, str);
        str = (char *) realloc(str, 25);
        strcat(str, "university");
        printf("String = %s, Address = %u\n", str, str);
```

```
        free(str);
        return(0);
}
```

(e) Pointer Comparison: pointers can be compared using equality and relational operators, but it is meaningless unless the pointer point to the element of the same array. It compares the addresses stored in the pointers. It can show that one pointer point to higher element of the array than the other pointer pointing to element of the same array. Common use of pointer comparison is determining whether pointer is NULL.

Q. No. 6. Write a program to produce the following output:

```
1
01
101
0101
10101
```

Ans:

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,j;
    for(i=1;i<=5;i++)
    {
     for(j=i;j>=1;j--)
     {
          printf("%d",j%2);
          count++;
     }
     printf("\n");
    }
     getch();
     return 0;
}
```

Q. No. 7. Write a program to compare two files specified by the user, displaying a message indicating whether the files are identical or different

Ans:

```
#include<stdio.h>
#include<conio.h>
main()
{
    FILE *fp1,*fp2;
    char ch1,ch2;
    fp1=fopen("file1.c","r");
    if(fp1==NULL)
    {
        printf("Problem in file opening");
        exit(0);
    }
    fp2=fopen("file2.c","r");
    if(fp2==NULL)
    {

        printf("Problem in file opening");
        fclose(fp1);
        exit(0);
    }
    ch1=fgetc(fp1);
    ch2=fgetc(fp2);
    while(ch1!=EOF || ch2!=EOF)
    {
        ch1=fgetc(fp1);
        ch2=fgetc(fp2);
        if(ch1!=ch2)
        break;
        printf("%c",ch1);
     }
    if(ch1!=ch2)
    {
       printf("\nBoth file are not same");
    }
    else
    {
       printf("\nBoth file are same");
    }
```

```
    fclose(fp1);
    fclose(fp2);
    getch();
}
```