Q. Differences between user defined function and library function.

SN	Library Function		User-defined function
i)	It is pre-defined function in a header file or pre-processor directory.	i)	It is not pre-defined function. It is defined by the programmer according to the need.
ii)	Programmer can simply use this function by including respective header file.	ii)	Programmer has to decleare define and use this function by themself.
iii)	The programme using library function will be usually short as the programmer does not have to define the function.	iii)	The programme user define function will be lengthy as the programmer has to define the function.
iv)	Programme development time will be faster.	iv)	Programme development time will be usually slower.
v)	Programme will be simple.	v)	Programme will be complex.
vi)	This function requires header file to use it.	vi)	This function requires function prototype to use it.

Recursion:-

Recursion is a full technique of writing a complicated algorithm in easy way. According to this technique a problem is defined in term of itself. A problem is solved by dividing it into smaller problem, which are similar in nature to the original problem. These smaller problems are solved and their solution are applied to get the final solution of our original problem.

In C it is possible for the function to call themselves. A function is called recursive if a statement within a body of a function calls the same function. Sometimes called circular defination recursion is the process of defining something in terms of itself. The recussion continues until some condition is meet to prevent it. A function will be recursive if it contains following features:

1) function should call itself
2) function shall have a stopping condition (base criteria) and every time the function calls itself it must be closer to base criteria.

Call by value & Call by refence:-

Call by value (Passing by value):-

In pass by value, values of variables are passed to the function from the calling function. This method copies the actual parameters into formal parameters. In other words, when the function is called, a seperate copy of the variables is created in the ~~mor~~ memory and value of orginal variables is given to these variables. So, if any changes are made in the value, it is not reflected to the orginal variable (of calling function). It can return only one value.

Example:-

```
#include<stdio.h>
#include<conio.h>
void main()
{  int a =15;
   clrscr();
printf("Before calling function, a=%d \n", a);
   change(a);
printf(" After calling function, a=%d \n", a);
   getch();
}
void change(int x)
{  x= x+5;
}
```

Output:

Before calling function, a = 15.
After calling function, a = 15.

ii) Call by refence (Passing by refrence):-

In passing by refrence we pass the address or location of a variable to the function during function call. Pointers are used to call a function by refrence. When a function is called by refrence, then the formal arguments becomes refrences to the actual arguments. This means that the called function does not create its own copy of values rather it refers to the original values only by refrence name. Thus, function works with original data and the changes are made in the original data itself. It can return more than 1 value at a time.

Example:-

```
#include<stdio.h>
#include<conio.h>
void change (int *);
void main()
{ int a = 15;
  clrscr();
printf("Before calling function, a=%d\n", a);
  change (&a);
printf("After calling function, a=%d\n", a);
  getch();
}
```

```
void change (int* x)
{
    *x = *x + 5;
}
```

Output:

Before calling function, a = 15
After calling function, a = 20

Unit-8

Structure and Union:

Structure:

Structure is collection of logically related data items grouped together under a single name. In structure the individual elements may differ in type, that's why we can regard structure as a heterogenous user defined data type. The data items enclosed within a structure are known as members.

Syntax for structure defination:

```
struct structure_name { data_type member1;
                        data_type member2;
                        :
                        data_type membern;
                      };
```

Initializing structure variable:

The syntax of initilizing structure variable is similar to that of arrays. All the values are given in curly braces and the number, order and type of these values should be same as in structure defination.

For Example:

```
struct student{ char name[15];
                int roll;
                float fee;
              };
struct student st={"Sonia", 23, 1450.50};
```

(*) Differences between structure and union:

S.N	Structure	S.N	Union
i)	Memory occupied by structure is sum of individual data type.	i)	Memory occupied by union is of highest dat_type of all.
ii)	It can take part in complex data structure.	ii)	It can not take part in complex data structure.
iii)	Keyword struct is used.	iii)	Keyword union is used.
iv)	Every element values are independent to eachother.	iv)	If any of the value of any element has been changed there is direct impact to the other elements value.
v)	Memory allocation of every element is independent to eachother so the memory allocation is sum of every element.	v)	Memory allocation is performed by sharing the memory with highest data type.
vi)	All the members can be accessed simultaneously	vi)	Only one member is active ata time. So only one member can be accessed at a time.
vii)	Syntax struct struct_name { data_type member1; data_type member2; : datatypemember n; };	vii)	Syntax union union_name { data_type member1; data_type member2; : data_type member n; };

Unit-9

Pointers

Pointer :-

A pointer is a variable that stores the address of another variable. The address of a data item is the address of its first storage location. The address of a data item is a pointer to the data item, and a variable that holds the ~~adder~~ address is called a pointer variable.

& → represents Address of

* → represents value at address of.

+ Functions of pointer / Uses of pointer

Dynamic Memory Allocation :-

The process of allocating memory at the time of execution is called dynamic memory allocation. Dynamic memory allocation is necessary to manage available memory. The <stdio.h> provides four functions that can be used to manage dynamic memory which are calloc(), malloc(), free() and realloc().

In static memory allocation we could not increase or decrease the size of memory during the execution of program. For example if we decleare an array of integers as

```
int arr[100];
```

Now, in this the size of array is fixed

int a; -> declaration of pointer
int *Pa; } -> initialization of pointer
Pa = &a }

during compile time, which may cause two problems as:

i) The number of values to be stored may be less than the size of array which will cause the wastage of memory.

ii) If we want to store more values than the given size of array than the size of array can't be increased.

So, to overcome these problems we should be able to allocate memory at runtime, which is dynamic memory allocation.

The allocation and released of memory space can be done with the help of some built-in-function (library function) whose prototypes are found in alloc.h and stdio.h header files.

Pointers play an important role in dynamic memory allocation because we can access the dynamically allocated memory only through pointers.

Functions used to manage dymanic memory allocation

i) Malloc() -> This function is used to allocate memory dynamically. The malloc function allocates a memory block of size n bytes.

Malloc allocates a continious block of memory, if enough continious memory is not available, then malloc returns null.

for eg. int *p = (int*) malloc(100 * (size of int));

ii) free() → We cause the program to give back the allocated block to free memory. The argument to free is any address that will return by a prior call to malloc. If free is applied to a location that has been freed before, a double free memory error may occur.

for e.g. free(studentptxt).

iii) calloc() → The calloc function is used to allocate multiple blocks of memory. It is somewhat similar to malloc except for two differences. The first one is that it takes two arguments. The first argument specify the number of blocks and second one specify the size of each block. The other difference is that the memory allocated by malloc contains garbage value, while memory allocated by calloc() is initialized to zero.

for e.g. int *p = (int*) calloc (5, sizeof(int));

iv) realloc() →

The function realloc is used to change the size of memory block. It alters the size of memory block without loosing the old data. This is known as reallocation of memory. This function

also takes two arguments, first is a pointer to the block of memory that was previously allocated by malloc() or calloc() and second one is the new size for that block.

for e.g. ptr = (int*) realloc (ptr, newsize);

i.e int *P = (int*) realloc (P, 5);

Unit-10

Files Handling in C

Opening of file:-

A file must be opened before any input output operation is performed on the file. The process of establishing a connection between the program and file for performing different kinds of operations is called opening of the file. A structure named FILE is defined in the file stdio.h that contains all information about the file like name, status, buffer size, current position, end of file status etc.

All these details are hidden from the programming and the operating system takes care of all these things. A file pointer is a pointer to a structure of type FILE. Whenever a file is opened, a structure of type FILE is associated with it and a file pointer that points to this structure identify this file.

For example: FILE *fp;

The function fopen() is used to open a file.

E.g: fp = fopen("path:\\ filename.txt", "file operation mode");

Closing of file:-

The file that was opened using fopen() function must be closed when no more operations are to be performed on it. After closing the file, connection between file and program is broken and operations like read, write can't be done.

On closing the file, all the buffers associated with it are written to the file. The buffers allocated by the system for the files are freed after the file is closed, so that these buffers can be available for other files.

Although all the files are closed automatically when the program terminates, but sometimes it may be necessary to close the file by using fclose() function.

File opening modes:-

The different file modes that can be used in opening files are as follows:-

1) "w" (write) → If the file doesn't exist then this mode creates a new file for writing, and if the file already exists then the previous data is erased and the new data entered is written to the file.

ii) "a" (append) → If the file does not exist it also creates a new file (~~and to append entered data at the end of existing data~~) and if the file exists then the new data entered is appended at the end of existing data.

iii) "r" (read) → This mode is used for opening an existing file for reading purpose. The file to be opend must exist and the previous data of the file is not erased.

iv) "w+" (write+read) → This mode is same as "w" mode but in this mode we can also read and modify the data. If the file doesn't exist then a new file is created and if the file exists then previous data is erased.

v) "r+" (read+write) → This mode is same as "r" mode but in this mode we can also write and modify existing data. The file to be opened must exist and the previous data of file is not erased. Since we can add new data and modify existing data so this mode is also called updata mode.

vi) "a+" (append+read) → This mode is same as the "a" mode but in this mode we can also read the data stored in the file. If the file does not exist a new

file is created and if the file already exists then new data is appended at the end of existing data. We can not modify existing data in this mode.

vii) "wb" → Binary file opened in write mode.

viii) "ab" → Binary file opened in append mode.

ix) "rb" → Binary file opened in read mode.

x) "wb+" → Create binary file for read/write.

xi) "rb+" → Open a binary file for read/write.

xii) "ab+" → Append a binary file for read/write.

Input / Output Functions:-

1) Character I/O:

a) fputc() → This function writes a character to the specified file at the current file position and then increments the file position pointer.

b) fgetc() → This function reads a single character from a given file and increments the file pointer.

2) String I/O

a) fputs() → This function writes the null terminated string pointed by given character pointer to a file.

b) fgets() → This function reads characters from a file and these characters are stored in the string pointed by a character pointer.

3) Integer I/O

a) putw() → This function writes an integer value to the file pointed to by file_pointer.

b) getw() → This function returns the integer value from the file associated with file_pointer.

4) Formatted I/O

a) fprintf() → This function is same as the printf() function but it writes formatted data into the file instead of the screen. This function has same parameters as in printf() but it has one additional parameter which is a pointer of FILE type, that points to the file to which the output is to be written.

b) fscanf() → This function is similar to the scanf() but it reads data from file instead of screen, so, it has also one more parameter which is a pointer of FILE type and it points to the file from which data will be read.

5) Block Read / Write

a) fwrite() → This function is used for writing an entire block to a given file.

b) fread() → This function is used to read an entire block from a given file.

Unit-11

Introduction to Graphics :-

Graphics Function:-

The function which is used to draw different shapes, display text in different fonts, change colour etc. in C language is called graphics function. Using functions of graphics.h we can make graphics programs, animations, projects and games. We can draw circles, lines, rectangles, bars and many more geometrical figures.

There are numerous graphics functions available in C some of which are as follows:-

1) Putpixel (x, y, colour):- The functionality of this function is to put a pixel or in other words a dot at position (x, y) given in inputed arguments. Here one must understand that the whole screen is imaged as a graph. In other words the pixel at the top left corner of the screen represents the value (0,0) i.e. origin. Here the ~~toulo~~ colour is the integer value associated with colours and when specified the picture elements or the dot.

ii) line (x_1, y_1, x_2, y_2):- The functionality of this function is to draw line from (x_1, y_1) to (x_2, y_2). Here also the co-ordinates are passed taking the pixel (0,0), at the top left corner of the screen as the origin.

iii) getpixel (x, y) :- This function when involved gets the colour of the pixel specified. The colour got will be the integer value associated with that colour and hence the function gets an integer value as return value. So, the smallest element on the graphics display screen is a pixel or a dot and the pixels are used in this way to place images in graphics screen in C programming languages.