UNIT V

FUNCTIONS AND POINTERS

Handling of Character Strings – User-defined Functions – Definitions – Declarations - Call by reference – Call by value – Structures and Unions – Pointers – Arrays – The Preprocessor – Developing a C Program : Some Guidelines.

2 MARKS

1. **What is meant by Recursive function?**
   If a function calls itself again and again, then that function is called Recursive function.
2. **What is an array?**
   An array is a group of similar data types stored under a common name.
   int a[10];
   Here a[10] is an array with 10 values.
3. **What is a Pointer? How a variable is declared to the pointer?** **(MAY 2009)**
   Pointer is a variable which holds the address of another variable.
   Pointer Declaration:
   datatype *variable-name;
   Example:
   int *x, c=5;
   x=&a;
4. **What are the uses of Pointers?**
   - Pointers are used to return more than one value to the function
   - Pointers are more efficient in handling the data in arrays
   - Pointers reduce the length and complexity of the program
   - They increase the execution speed
   - The pointers saves data storage space in memory
5. **What is the output of the program?**

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

   Output:
   1. 2
   2.
6. **What are * and & operators means?**
   '*' operator means 'value at the address'
   '&' operator means 'address of'
7. **What is meant by Preprocessor?**
   Preprocessor is the program, that process our source program before the compilation.

8. **How can you return more than one value from a function?**

   A Function returns only one value. By using pointer we can return more than one value.

9. **Is it possible to place a return statement anywhere in 'C' program?**

   Yes. The return statement can occur anywhere.

10. **What are the main elements of an array declaration?**
    - Array name
    - Type and
    - Size

11. **List the header files in 'C' language.**

    <stdio.h> contains standard I/O functions
    <ctype.h> contains character handling functions
    <stdlib.h> contains general utility functions
    <string.h> contains string manipulation functions
    <math.h> contains mathematical functions
    <time.h> contains time manipulation functions

12. **What are the steps involved in program development life cycle?**
    1. Program Design
    2. Program Coding
    3. Program Testing & Debugging

13. **What are the types of errors occurred in C program?**
    1. Syntax errors
    2. Runtime errors
    3. Logical errors
    4. Latent errors

14. **What is testing?**

    Testing is the process of executing the program with sample or tested data.

15. **What are the types of testing?**
    - Human testing
    - Computer based testing

16. **How do you define enumerated data type?**

```
enum mar_status
{   single,married,widow        };
enum mar_status person1,person2;
person1=married;
```

    Here the person1 is assigned to value zero.

17. **What is meant by debugging?**

    Debugging is the process of locating and isolating the errors.

18. **Specify any five syntax error messages.**
    - Missing semicolon
    - Missing braces
    - Missing quotes
    - Improper comment characters
    - Undeclared variables

19. **What are the pre-processor directives?**
    - Macro Inclusion
    - Conditional Inclusion
    - File Inclusion

20. What is dynamic memory allocation?

Allocating the memory at run time is called as dynamic memory allocation.

21. What are the various dynamic memory allocation functions?

malloc() - Used to allocate blocks of memory in required size of bytes.
free() - Used to release previously allocated memory space.
calloc() - Used to allocate memory space for an array of elements.
realloac() - Used to modify the size of the previously allocated memory space.

22. What is the deference between declaring a variable and defining a variable?

- Declaring a variable means describing its type to the compiler but not allocating any space for it.
- Defining a variable means declaring it and also allocating space to hold the variable. A variable
- Can also be initialized at the time it is defined. To put it simply, a declaration says to the compiler,
- "Some where in the program there will be a variable with this name, and this is the kind of data
- Type it is." On the other hand, a definition says, "Right here is this variable with this name and
- This data type". Note that a variable can be declared any number of times, but it must be defied
- Exactly once. For this reason, definitions do not belong in header files, where they might get #included into more than one place in a program.

23. Why does n++ execute than n=n+1?

The expression n++ requires a single machine instruction such as INR to carry out the increment operation whereas; n+1 requires more instructions to carry out this operation.

24. Why is it necessary to give the size of an array in an array declaration?

When an array is declared, the compiler allocates a base address and reserves enough space in the memory for all the elements of the array. The size is required to allocate the required space. Thus, the size must be mentioned.

25. Where in memory are variables stored?

Variables can be stored in several places in memory, depending on their lifetime.

(1) Variables that are defined outside any function (whether of global or file static scope), and variables that are defined inside a function as static variables, exist for the lifetime of the program's execution. These variables are stored in the data segment. The data segment is a fixed-size area in memory set aside for these variables.

(2) Variables that are the arguments functions exist only during the execution of that function. These variables are stored on the stack. The stack is an area of memory that starts out as small and
grows automatically up to some predefined limit.

(3) The third area is the one that does not actually store variables but can be used to store data pointed to by variables. Pointer variables that are assigned to the result of a call to the function malloc() contain the address of a dynamically allocated area of memory. This memory is in an area called the heap.

26. What is an heap memory?

The heap is another area that starts out as small and grows, but it grows only when the programmer explicitly calls malloc() or other memory allocation functions, such as calloc(). The heap can share a memory segment with either the data segment or the stack, or

it can have its own segment, it all depends on the compiler options and operating system. The heap, like the stack, has a limit on how much it can grow, and the same rules apply as to how that limit is determined.

27. What is the difference between an array and pointer?

Difference between arrays and pointers are as follows.

Array	Pointer
1.Array allocates space automatically. 2.It cannot be resized. 3.It cannot be reassigned. 4.Size of(array name) gives the number of bytes occupied by the array.	1.Pointer is explicitly assigned to point to an allocated space. 2.It can be resized using realloc (). 3.Pointers can be reassigned. 4.Sezeof(pointer name) returns the number of bytes used to store the pointer variable.

27. What is the purpose of the function main()? (MAY 2009)

The function main () invokes other functions within it. It is the first function to be
called when the program starts execution.

Some salient points about main() are as follows:

1. It is the starting function .
2. It returns an int value to the environment that called the program.
3. Recursive call is allowed for main() also.
4. It is a user-defined function.
5. Program exection ends when the closing brace of the function main() is reached.
6. It has two arguments (a) argument count and (b)argument vector (reprensents strings passed.)
7. Any user-defined name can also be used as parameters for main() instead of argc and argv

28. What is dangling pointer?

In C, a pointer may be used to hold the address of dynamically allocated memory.

After this memory is freed with the free() function, the pointer itself will still contain the address of the released block. This is referred to as a dangling pointer. Using the pointer in this state is a serious programming error. Pointer should be assigned NULL after freeing memory to avoid this bug.

29. Compare arrays and structures.

Comparison of arrays and structures is as follows.

Arrays	Structures
An array is a collection of data items of same data type. Arrays can only be declared. There is no keyword for arrays. An array name represents the address of the starting element. An array cannot have bit fields.	A structure is a collection of data items of different data types. Structures can be declared and defined. The deyword for structures is struct. A structrure name is known as tag. It is a shorthand notation of the declaration. A structure may contain bit fields.

30. Compare structures and unions.

Structure	Union
Every member has its own memory.	All members use the same memory.
The keyword used is struct.	The keyword used is union.
All members occupy separate memory location, hence different interpretations of the same memory location are not possible.	Different interpretations for the same memory location are possible.
Consumes more space compared to union.	Conservation of memory is possible.

31. Is it better to use a macro or a function?

Macros are more efficient (and faster) than function, because their corresponding code is inserted directly at the point where the macro is called. There is no overhead involved in using a macro like there is in placing a call to a function.

However, macros are generally small and cannot handle large, complex coding constructs. In cases where large, complex constructs are to handled, functions are more suited, additionally; macros are expanded inline, which means that the code is replicated for each occurrence of a macro.

32. List the characteristics of Arrays.

All elements of an array share the same name, and they are distinguished form one another with help of an element number.

Any particular element of an array can be modified separately without disturbing other elements.

33. What are the types of Arrays?

1.One-Dimensional Array
2. Two-Dimensional Array
3. Multi-Dimensional Array

34. What is the use of '\0' character?

When declaring character arrays (strings), '\0' (NULL) character is automatically added at end. The '\0' character acts as an end of character array.

35. Define sscanf() and sprint() functions.

The sscanf():

This function allows to read character from a character Array and writes to another array. Similar to scanf(), but instead of reading from standard input, it reads from an array.

The sprintf():

This function writes the values of any data type to an array of characters.

36. Define Strings.

Strings:

The group of characters, digit and symnbols enclosed within quotes is called as Stirng (or) character Arrays. Strings are always terminated with '\0' (NULL) character. The compiler automatically adds '\0' at the end of the strings.

Example:

char name[]={'C','O','L','L','E','G','E','E','\0'};

The character of a string are stored in contiguous memory locations as follows:

C	O	L	L	E	G	E	\0
1000	1001	1002	1003	1004	1005	1006	1007

37. What is the use of 'typedef''?

It is used to create a new data using the existing type.

Syntax: typedef data type name;

Example:

typedef int hours: hours hrs;/* Now, hours can be used as new datatype */

38. What is 'C' functions? Why they are used?

A function is a self-contained block (or) a sub-program of one or more statements that performs a special task when called. To perform a task repetitively then it is not necessary to re-write the particular block of the program again and again. The function defined can be used for any number of times to perform the task.

39. Differentiate library functions and User-defined functions.

Library Functions	**User-defined Functions**
a) Library functions are pre-defined set of functions that are defined in C libraries. b) User can only use the function but cannot change (or) modify this function.	a) The User-defined functions are the functions defined by the user according to his/her requirement. b) User can use this type of function. User can also modify this function.

40. What are the steps in writing a function in a program.

a) Function Declaration (Prototype declaration):
   Every user-defined functions has to be declared before the main().
b) Function Callings:
   The user-defined functions can be called inside any functions like main(), user-defined function, etc.
c) Function Definition:
   The function definition block is used to define the user-defined functions with statements.

41. What is a use of 'return' Keyword?

The 'return' Keyword is used only when a function returns a value.

42. Give the syntax for using user-defined functions in a program.

Syntax for using user-defined functions in a program

Syntax:

```
    function declaration;                          function definition;
     main()                                              main()
     {                                             {
     ======                                        ======
function calling;              (or)                function calling;
     ======                                        ======
}                                                  }
function definition;
```

43. Classify the functions based on arguments and return values.

Depending on the arguments and return values, functions are classified into four types.

a) Function without arguments and return values.
b) Function with arguments but without return values.
c) Function without arguments but with return values.
d) Function with arguments and return values.

44. Distinguish between Call by value Call by reference.

Call by value	**Call by reference.**
a) In call by value, the value of actual agreements is passed to the formal arguments and the operation is done on formal arguments. b) Formal arguments values are photocopies of actual arguments values. c) Changes made in formal arguments valued do not affect the actual arguments values.	a) In call by reference, the address of actual argurment values is passed to formal argument values. b) Formal arguments values are pointers to the actual argument values. c) Since Address is passed, the changes made in the both arguments values are permanent.

12 MARKS

1. **What are functions? Explain the types of functions in detail with an example program for each type.**

A function is a self contained block or a sub program of one or more statements that performs a special task when called.

Types:

- Library Functions
- User Defined functions

(a) Function Declaration

returntype function-name(Parameters);

Example:

int square(int, int);

(b) Function calling

function-name(actual parameters);

Example:

int square(a,b);

(c) Function Definition:

```
returntype function-name(formal parameters)
        {
                local variable declaration;
                statement 1;
                statement 2;
                return(value);
        }
```

Example:

```
void square(int a, int b)
{
printf("%d",(a*b));
}
```

Example for functions:

- Addition of two numbers where addition is a separate function

- Program using function for evaluating Fibonacci series.

2. Define arrays. Explain the array types with an example program for each type.

Arrays are data structures which hold multiple variables of the same data type. Consider the case where a programmer needs to keep track of a number of people within an organization. So far, our initial attempt will be to create a specific variable for each user.

This might look like,

```
int name1 = 101;
int name2 = 232;
int name3 = 231;
```

It becomes increasingly more difficult to keep track of this as the number of variables increase. Arrays offer a solution to this problem. An array is a multi-element box, a bit like a filing cabinet, and uses an indexing system to find each variable stored within it. In C, indexing starts at **zero**. Arrays, like other variables in C, must be declared before they can be used. The replacement of the above example using arrays looks like,

```
int names[4];
names[0] = 101;
names[1] = 232;
names[2] = 231;
names[3] = 0;
```

We created an array called names, which has space for four integer variables. You may also see that we stored 0 in the last space of the array. This is a common technique used by C programmers to signify the end of an array. Arrays have the following syntax, using square brackets to access each indexed value (called an element).

x[i]

so that x[5] refers to the sixth element in an array called x. In C, array elements start with 0. Assigning values to array elements is done by,

```
x[10] = g; and assigning array elements to a variable is done by,
g = x[10];
```

In the following example, a character based array named word is declared, and each element is assigned a character. The last element is filled with a zero value, to signify the end of the character string (in C, there is no string type, so character based arrays are used to hold strings). A printf statement is then used to print out all elements of the array.

```
/* Introducing array's, 2 */
#include <stdio.h>
main()
{
char word[20];
word[0] = 'H';
word[1] = 'e';
word[2] = 'l';
word[3] = 'l';
word[4] = 'o';
word[5] = 0;
```

```
printf("The contents of word[] is -->%s\n", word );
}
```

DECLARING ARRAYS

Arrays may consist of any of the valid data types. Arrays are declared along with all other variables in the declaration section of the program.

```
/* Introducing array's */
#include <stdio.h>
main()
{
int numbers[100];
float averages[20];
numbers[2] = 10;
--numbers[2];
printf("The 3rd element of array numbers is %d\n", numbers[2]);
}
```

The above program declares two arrays, assigns 10 to the value of the 3rd element of array numbers, decrements this value (--numbers[2]), and finally prints the value. The number of elements that each array is to have is included inside the square brackets

ASSIGNING INITIAL VALUES TO ARRAYS

The declaration is preceded by the word static. The initial values are enclosed in braces,

Example:

```
#include <stdio.h>
main()
{
int x;
static int values[] = { 1,2,3,4,5,6,7,8,9 };
static char word[] = { 'H','e','l','l','o' };
for( x = 0; x < 9; ++x )
printf("Values [%d] is %d\n", x, values[x]);
}
```

MULTI DIMENSIONED ARRAYS

Multi-dimensioned arrays have two or more index values which specify the element in the array.

```
multi[i][j];
```

In the above example, the first index value i specifies a row index, whilst j specifies a column index.

DECLARATION

```
int m1[10][10];
static int m2[2][2] = { {0,1}, {2,3} };
sum = m1[i][j] + m2[k][l];
```

NOTE the strange way that the initial values have been assigned to the two-dimensional array m2. Inside the braces are,

{ 0, 1 },
{ 2, 3 }

Remember that arrays are split up into row and columns. The first is the row, the second is the column. Looking at the initial values assigned to m2, they are,

m2[0][0] = 0
m2[0][1] = 1
m2[1][0] = 2
m2[1][1] = 3

Example:

```
#include <stdio.h>
main()
{
static int m[][] = { {10,5,-3}, {9, 0, 0}, {32,20,1}, {0,0,8} };
int row, column, sum;
sum = 0;
for( row = 0; row < 4; row++ )
for( column = 0; column < 3; column++ )
sum = sum + m[row][column];
printf("The total is %d\n", sum );
}
```

CHARACTER ARRAYS [STRINGS]

Consider the following program,

```
#include <stdio.h>
main()
{
static char name1[] = {'H','e','l','l','o'};
static char name2[] = "Hello";
printf("%s\n", name1);
printf("%s\n", name2);
}
```

The difference between the two arrays is that name2 has a null placed at the end of the string, ie, in name2[5], whilst name1 has not. To insert a null at the end of the name1 array, the initialization can be changed to,

```
static char name1[] = {'H','e','l','l','o','\0'};
```

Consider the following program, which initialises the contents of the character based array word during the program, using the function strcpy, which necessitates using the include file string.h

Example:

```
#include <stdio.h>
#include <string.h>
main()
{
char word[20];
strcpy( word, "hi there." );
```

```
printf("%s\n", word );
}
```

3. **Explain the standard string functions with example to support each type.**

Strings:

The group of characters, digits and symbols enclosed within quotes is called as strings or character arrays. Strings are always terminated with '\0' character(NULL).

Example:

char name[ ] = {'H','E','L','L','O'};

Standard String Functions:

- strlen( )
- strcpy( )
- strncpy( )
- stricmp( )
- strcmp( )
- strncmp( )
- strcat( )
- strrev( ) etc.,

Example program:

- To read and display a string.
- Program to count the number of lines, words and characters in a text.

4. **What are pointers? When and why they are used? Explain in detail with sample programs. (JAN 2009/MAY 2009)**

Pointer variable is needed to store the memory address of any variable. Denoted by (*) asterisk.

Pointer Declaration:

Syntax:

datatype *variable-name;

Exmaple:

int *a;

- Pointers and Arrays
- Pointers and Strings
- Pointer as function arguments
- Pointer too pointer

Example program:

- To add two numbers through variables and their pointers.
- To assign a pointer value to another variable.

5. **Describe in detail about the Preprocessors in C. (MAY 2009)**

THE PREPROCESSOR

The define statement is used to make programs more readable, and allow the inclusion of macros. Consider the following examples,

```
#define TRUE 1 /* Do not use a semi-colon , # must be first character on line */
#define FALSE 0
#define NULL 0
#define AND &
#define OR |
```

```
#define EQUALS ==
game_over = TRUE;
while( list_pointer != NULL )
.....
```

MACROS

Macros are inline code which are substituted at compile time. The definition of a macro, which accepts an argument when referenced,

```
#define SQUARE(x) (x)*(x)
y = SQUARE(v);
```

In this case, v is equated with x in the macro definition of square, so the variable y is assigned the square of v. The brackets in the macro definition of square are necessary for correct evaluation.

The expansion of the macro becomes

```
y = (v) * (v);
```

Naturally, macro definitions can also contain other macro definitions,

```
#define IS_LOWERCASE(x) (( (x)>='a') && ( (x) <='z') )
#define TO_UPPERCASE(x) (IS_LOWERCASE (x)?(x)-'a'+'A':(x))
while(*string) {
*string = TO_UPPERCASE (*string);
++string;
}
```

CONDITIONAL COMPILATIONS

These are used to direct the compiler to compile/or not compile the lines that follow

```
#ifdef NULL
#define NL 10
#define SP 32
#endif
```

In the preceding case, the definition of NL and SP will only occur if NULL has been defined prior to the compiler encountering the #ifdef NULL statement. The scope of a definition may be limited by

```
#undef NULL
```

This renders the identification of NULL invalid from that point onwards in the source file.

Typedef

This statement is used to classify existing C data types, eg,

```
typedef int counter; /* redefines counter as an integer */
counter j, n; /* counter now used to define j and n as integers */
typedef struct {
int month, day, year;
} DATE;
DATE todays_date; /* same as struct date todays_date */
```

ENUMERATED DATA TYPES

Enumerated data type variables can only assume values which have been previously declared.

```
enum month { jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec };
enum month this_month;
this_month = feb;
```

In the above declaration, month is declared as an enumerated data type. It consists of a set of values, jan to dec. Numerically, jan is given the value 1, feb the value 2, and so on. The variable this_month is declared to be of the same type as month, then is assigned the value associated with feb. This_month cannot be assigned any values outside those specified in the initialization list for the declaration of month.

Example:

```
#include <stdio.h>
main()
{
char *pwest = "west",*pnorth = "north", *peast="east", *psouth = "south";
enum location { east=1, west=2, south=3, north=4};
enum location direction;
direction = east;
if( direction == east )
printf("Cannot go %s\n", peast);
}
```

The variables defined in the enumerated variable location should be assigned initial values.

DECLARING VARIABLES TO BE REGISTER BASED

Some routines may be time or space critical. Variables can be defined as being register based by the following declaration,

```
register int index;
```

DECLARING VARIABLES TO BE EXTERNAL

Here variables may exist in separately compiled modules, and to declare that the variable is external,

```
extern int move_number;
```

This means that the data storage for the variable move_number resides in another source module, which will be linked with this module to form an executable program. In using a variable across a number of independently compiled modules, space should be allocated in only one module, whilst all other modules use the extern directive to access the variable.

NULL STATEMENTS

These are statements which do not have any body associated with them.

```
/* sums all integers in array a containing n elements and initializes */
/* two variables at the start of the for loop */
for( sum = 0, i = 0; i < n; sum += a[i++] )
;
/* Copies characters from standard input to standard output until EOF is reached */
for( ; (c = getchar ()) != EOF; putchar (c));
```

COMMAND LINE ARGUMENTS

It is possible to pass arguments to C programs when they are executed. The brackets which follow main are used for this purpose. argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument which is passed to main. A simple example follows, which checks to see if a single argument is supplied on the command line when the program is invoked.

```
#include <stdio.h>
main( int argc, char *argv[] )
{
if( argc == 2 )
printf("The argument supplied is %s\n", argv[1]);
else if( argc > 2 )
printf("Too many arguments supplied.\n");
else
printf("One argument expected.\n");
}
```

Note that *argv[0] is the name of the program invoked, which means that *argv[1] is a pointer to the first argument supplied, and *argv[n] is the last argument. If no arguments are supplied, argc will be one. Thus for n arguments, argc will be equal to n + 1. The program is called by the command line, myprog argument1.

6. Brief call by value and call by reference in detail. (MAY 2009)

Call by value:

In call by value the value of the actual arguments are passed to the formal arguments and the operation is done on formal arguments.

Example program:

- To send two integer values using “call by value”.

Call by reference:

In call by reference the address of actual argument values are passed to formal argument values.

Example program:

- To send a value by reference to user defined function.

7. Discuss about function prototypes in detail.

(or)

Explain about the different parameter passing methods with examples (JAN 2009)

- Function with arguments and return type.
- Function without arguments and return type.
- Function with arguments and no return type.
- Function without arguments and return type.

8. Define Structures. Explain structures in detail. (JAN 2009 / MAY 2009)

A structure is a collection of one or more variables of different data types grouped together under a single name. It contains different data types.

Syntax:

```
struct struct-name
{
type variable 1;
type variable 2;
type variable n;
} structure_variables;
```

Example:

```
struct student
{
char name[25];
int rollno;
int m1,m2,m3,total;
float avg;
}s1,s2;
```

- Structure within structure
- Array of structures
- Pointers to structures
- Structures and functions

Example program:

- To define a structure and read the member variable values from user.
- To copy structure elements from one object to another object.

9. Define Union. Explain Union in detail. (JAN 2009)

Union is a collection of variables similar to structure. The union requires bytes that are equal to number of bytes required for the largest number.

Example:

```
union student
{
char name[20];
int rollno,m1,m2,m3,tot;
float avg;
}s1;
```

Union of structure

Union can be nested with another union.

Example program:

- Program to use structure within union. Display the contents of structure elements.

****