Set - 2022

Group A (2 x 12 = 24)

Answer Two Questions

1.a. Define debugging and testing. [2]

ans. Testing is the process using which we find errors and bugs. Debugging is the process using which we correct the bugs that we found during the testing process.

b. What do you mean by brancing and looping? What is escape sequence? Explain different types of operators in C with examples. [2+2+6]

ans. Branching is also known as selective control structure in which selection is made on the basis of given condition. When condition is true the program will go towards certain statements & otherway when the condition is false. It can be classified into two types:

a. Conditional Statement.
b. Switch case

A looping is a repetitive control structure which execute a block of program statements repeatedly for specified number of time or till the given condition is satisfied. There are mainly three types of loop:

a. While loop
b. do while loop
c. For loop

- Escape sequences are character combinations consisting of a backslash (\) followed by a letter or by a combination of digits. For eg '\n' represent new line.

- An operator is a symbol that helps to perform certain mathematical calculations and logical manipulation. Eg: +, -, *, ::, =, >, c. The data item that operator acts upon is called operands. Operators can be classified into two types: on the basis of operands
: on the basis of operation

On the basis of operands
a. Unary Operator
b. Binary Operator
c. Ternary Operator

a. Unary Operator:
- The operator which use only one operand for operation is called unary operator. Eg: increment and decrement i.e. a++, --a, ++a etc.

b. Binary Operator:
- The operator which use two operand for operation is called binary operator. Eg: +, -, =, <, > i.e. 2+2, 2-2.

c. Ternary Operator:
- It is also known as conditional operator. It requires three operands it evaluates the condition and then assigned the required expression.

Syntax:

Condition ? Expression 1 : Expression 2;

If condition is true then expression 1 is evaluated and this becomes the value of the conditional expression and if condition is false, then expression 2 is evaluated.

On the basis of operation

1. Arithmetic operator
2. Relational operator
3. logical operator or Boolean operator
4. Assignment operator
5. Increment & Decrement operator
6. Conditional Operator
7. Comma Operator
8. Size of operator

1. Arithmatic operator:

- It is used for arithmatic operation like addition, substraction, multiplication & division.

Operator	Meaning	Example
+	Addition operator	a+b
-	Subtraction operator	a-b
*	Multiplication operator	a*b
/	Division operator	a/b
%	Modulus operator	a%b

2. Relational operator:

- Operators that are used for comparing relation between operands. In c-programming we can compare the variables using the relational operators. It is also called comparison operator as it is used to compare any two expression.

Operator	Meaning	Example
<	is less than	a<b
<=	less than equal to	a<=b
>=	greater than equal to	a>=b
>	greater than	a>b
==	equal to	a==b
!=	Not equal to	a!=b

3. Logical operator or Boolean operator:

- Logical operator are used for combining logical expressions which are finally evaluated as true or false according to operator use. logical operators are used for logical decision making in c-programming.

Operator	Meaning	Example
&&	Logical AND	(a>b)&&(a>c)
!! or ||	logical OR	(a>b)!!(a>c)
!	logical NOT	!(a==b)

4. Assignment Operator:

- It is used for assigning a value or a result of an expression to an identifier. It is the combination of arithmetic operator & assignment operator & performs

the task. They are also known as short hand operator.

Operators	Statement with short hand operator	Statement with simple assignment
+:	a+=1	a=a+1
-:	a-=1	a=a-1
*:	a*=1	a=a*1
/:	a/=2	a=a/2
%:	a%=b	a=a%b

Here, Arithmetic operator performs it's operation first and their assignment operators assigns the value.

5. Increment / Decrement operators:

- These operators are also known as unary operator which performs operation upon one operand. There are two types of numeric operator.

a. Prefix : ++ variables, -- variables;

b. Postfix: variables ++, variable --;

4. eg(i) m=5

y=++m;

The value of y & m would be 6.

eg(ii) m=5;

y=m++;

The value of y is 5 & m is 6.

a. Prefix Operator:

- It first adds 1 to be operand & then result is assigned to the variable on the left.

b. Postfix operator:

- It assigns the value to the variable on the left first and then increases or decreases the operand.

6. Conditional Operator (Ternary): (? :)

- Conditional Operator requires three operands. It evaluates the condition & then assigns the required expressions.

Condition-1 ? Expression 1 : Expression 2

If condition 1 is true then expression 1 is evaluated and this becomes the value of the conditional expression & if condition 1 is false, then expression 2 is evaluated.

```
eg: a = 5;
    b = 9;
    x = (a>b)?a:b;
```

7. Comma Operator:

- It is used to link related expression together. It evaluates the expression from left to right and the value of right post expression is the value of combined expression.

```
res=(num1 , num2);
```

seperator: In the functional call declaring variable etc.

```
eg: int num1 =10, num2 = 20;
```

8. Size of operator( ):

- The size of operator gives the size of the data type of variable in terms of bytes occupied in memory.

Eg: int k;
n = sizeof(k);

char n;
m = size of(n);

2.a. Find the output that will be generated by the following program: [6]

```
#include<stdio.h>
int main( )
{
    int a=6, b=2, x=0;
    while(--a>0 && b++<=7){
        if(a%2==0){
            x+=b/2;
        }
    }
    printf("\n a= %d", a);
    printf("\n b= %d", b);
    printf("\n x= %d", x);
    return 0;
}
```

Output:

a = 0
b = 7
c = 5

b. Write a program to delete white space and vowels from an ~~out~~ input sentences using pointer. [6]

```
#include <stdio.h>
int main( )
{
    char sen[100], *ptr, temp[100], *p;
    int i=0, j=0;
    ptr = sen;
    p = temp;
    puts("\nEnter your sentences: ");
    gets(ptr);
    for(i=0; *(ptr+i)!='\0'; i++)
    {
        if(*(ptr+i)==32)
        {
            continue;
        }
        else if(*(ptr+i)=='A'||*(ptr+i)=='E'||*(ptr+i)=='I'||
        *(ptr+i)=='O'||*(ptr+i)=='U'||*(ptr+i)=='a'||*(ptr+i)=='e'
        ||*(ptr+i)=='i'||*(ptr+i)=='o'||*(ptr+i)=='u')
        {
            continue;
        }
        else
        {
            *(p+j) = *(ptr+i);
            j++;
        }
```

```
}
*(ptr+j) = '\0';
printf("%s", temp);
return 0;
}
```

3. Explain different data types in C. Write a program to initialize an integer array of size 20 and arrange the elements in ascending order. (Use user-defined function). [4 + 8].

ans. A data type is the type of data use in the program. It is the storage representation and machine instruction to handle constant and variable. There are four kind of data types: They are: primary data type
: user-defined data type
: derived data type
: Empty data type

a. Primary data type:

- Primary data type is such data type which doesn't use any modifiers. It can be characterized as follow:

i. Integers

signed	unsigned
int	unsigned int
short	unsigned short int
long int	unsigned long int

eg: int a=5

In signed the number can be positive or negative in result. Whereas, in unsigned the number will be positive result.

ii. Floating point:

Float, Double float, long double

eg: float b=5.6

iii. character:

eg: char a=';'

char a[]="Nepal"

b. User-defined datatype:

- The user defined data types can be later used to declare variable. User defined data types define using type data and enum.

As an example let us consider how we can define a new type named "Letter". This letter, can then be used in the same way as the predetermined datatype of c.

Syntax: <type def><data type><new type>;

<new type><variable;

eg: type def int letter;

letter a;

a=5;

Here, a new type letter is created whose data type is integer. Later this letter is used as data type to declare the variable name & address.

c. Derived data type:

- Different user defined data type can be created using fundamental data types. Theyo are array, union, structure, function are derived data types.

d. Empty data-type:

- Empty /Void data-type is an empty data type that has no value and no operations. It can be used in functions & pointers.

```
//Arranging integer in ascending order size of 20
//finding greatest integer in an array using function.
#include<stdio.h>
void ascending (int [ ]);
int main( )
{
    int num[20], i=0;
    for(i=0 ; i<num ; i++)
    {
        printf("\nEnter number [%d]: ", i+1);
        scanf ("%d", &num[i]);
    }
    ascending (num);
    return 0;
}
void ascending (int n[ ])
{
    int temp, j=0, k=0;
```

```
for(k=0 ; k<20 ; k++)
{
    for(j=k+1 ; j<20 ; j++)
    {
        if(n[k]>n[j])
        {
            temp=n[k];
            n[k]=n[j];
            n[j]=temp;
        }
    }
}
printf("\nThe arranged form are:\n");
for(j=0;j<20;j++)
{
    printf("%d\t",n[j]);
}
}
```

9. What do you mean by functions in C? Differentiate between passing by value and passing by reference with example. [2+6]

ans. A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions. In order to enable a function in c program, the program must have three parts: a. Function declaration
b. Function call
c. Function defination

Syntax:

```
#include<stdio.h>
function declaration;
int main()
{
    ----------;
    function call;
    return 0;
}
function defination
{
    ----------;
    ----------;
}
```

S.No.	Passing by value	Passing by reference
1.	A copy of actual parameters is passed into formal parameters.	Reference of actual parameters is passed into formal parameters
2.	Changes in formal parameters will not result in changes in actual parameters.	Changes in formal parameters will result in changes in actual parameters.
3.	Seperate memory location is allocated for actual and formal parameters.	Same memory location is allocated for actual and formal parameters.
4.	Syntax: function_name(var_n1, var_n2)	Syntax: function_name(&var_n1, &var_n2)
5.	Primitive type are passed using "call_by_value".	Objects are implicitly passed using "call_by_reference".

6.	Example:
	#include <stdio.h>
	void swap(int, int);
	int main()
	{
	int a=0, b=0;
	printf("Enter 2 num:");
	scanf("%d %d", &a, &b);
	swap(a, b);
	return 0;
	}
		Example:
		#include <stdio.h>
		void swap(int *, int *);
		int main()
		{
		int a=0, b=0;
		printf("Enter 2 num:");
		scanf("%d %d", &a, &b);
		swap(&a, &b);
		return 0;
		}

```
void swap(int m, int n)
{
    int c = 0;
    c = m;
    m = n;
    n = c;
    printf("\n New value 1st is
      %d", m);
    printf("\n New value of 2nd is
      %d", n);
}
```

```
void swap (int *m, int *n)
{
    int z = 0;
    int * temp;
    temp = &z;
    * temp = *m;
    *m = *n;
    *n = * temp;
    printf("\n New value of 1 is %d,
      *m.);
    printf("\n New value of 2nd
      is %d", *n);
}
```

5. What do you mean by recursion? Write a program to calculate and display Fibonacci series up to n terms using recursion. [2+6]

ans. If a function calls itself again & again till certain condition meets then this phenomenon is known as recursion in C programming.

Syntax:

```
void recursion() {
    recursion(); /* function calls
}
int main() {
    recursion();
}
```

// Creating and displaying fibonacci series upto n^{th} term.

```
#include<stdio.h>
int fibo(int);
int main()
{
    int i=0, result=0, num=0;
    printf("\n Enter how many fibonacci series you want: ");
    scanf("%d", &num);
    for(i=0; i<num; i++)
    {
        result = fibo(i);
        printf("%d\t", result);
    }
    return 0;
}
int fibo(int n);
{
    if(n==0)
    {
        return 0;
    }
    else if(n==1)
    {
        return 1;
    }
    else
    {
        return fib(n-1)+ fib(n-2);
    }
}
```

6. A marking scheme in any exam is as follow:

Percentage (%)	Division
80 - 100	Distinction
60 - 79	First Division
50 - 59	Second Division
40 - 49	Third Division
0 - 39	Fail

Write a program that asks student to enter his/her marks in three subjects, calculate percentage and print the equivalent devision; otherwise, the division will be fail. Assume full marks and pass marks for each subject 100 and 40 respectively. Assume appropriate data types. [8]

```
#include<stdio.h>
int main( )
{
    float math=0, science=0, english=0, total=0;
    printf("\n Enter your marks on math, science and english
      respectively: ");
    scanf("%f %f %f", &math, &science, &english);
    total = (english + math + science) / 3;
    if (english>39 && science>39 && math>39)
    {
        if(total>=80 && total<=100)
        {
            printf("\n You got distinction by %.2f percentage", total);
        }
```

```
else if(total>=60 && total<=79)
{
    printf("\n You got first division by %.2f
            percentage", total);
}
else if(total>=50 && total<=59)
{
    printf("\n You got second division by %.2f
            percentage", total);
}
else
{
    printf("\n You got third division by %.2f percentage",
                                        total);
}
}
else
{
    printf("\n You become fail");
}
return 0;
}
```

7. Write a program to input a 4x3 matrix, then calculate its transpose matrix. [8]

```
#include<stdio.h>
int main()
{
    int a[4][3], b[3][4], i=0, j=0;
    printf("\nEnter matrix elements by row");
    for(i=0; i<4; i++)
    {
        for(j=0; j<3; j++)
        {
            printf("\nEnter for a[%d][%d]: ", i, j);
            scanf("%d", &a[i][j]);
            b[j][i] = a[i][j];
        }
    }
    printf("\nThe transpose form is: \n");
    for(i=0; i<3; i++)
    {
        for(j=0; j<4; j++)
        {
            printf("%d\t", b[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

8. What are the differences between structure and union? Explain passing structure to function with example. [3+5]

ans.

Parameter	Structure	Union
Keyword	A user can deploy the keyword struct to define a structure.	A user can deploy the keyword union to define a union.
Internal Implementation	The implementation of structure in C occurs internally because it contains seperate memory locations allotted to every input members.	In the case of a Union, the memory allocation occurs for only one member with the largest size among all the input variables. It shares the same location among all these members / objects.
Accessing Members	A user can access individual members at a given time.	A user can access only one member at a given time.
Syntax	Syntax: struct [structure name] { type element_1; type element_2; } variable_1, variable_2;	Syntax: union [union name] { type element_1; type element_2; } variable_1, variable_2;
Initialization	In the case of a structure, A user can initialize multiple members at the	In the case of a union, A user can only initiate the first member at a

	same time.	time.
Value Altering	Altering the values of a single member does not affect the other members of a structure.	When you alter the values of a single member, it affects the values of other members.

- Passing a structure to a function in C involves passing the structure's variable to the function as an argument. The structure variable is then accessed with the function using the dot operator to access its members.

Syntax:

```
struct structure_name{
    //member variables
};
void function_name(struct structure_name);~~variable_name)~~
int main()
{
    struct structure_name variable_name;
    //initialize structure variable
    function_name(variable_name);
    return 0;
}
void function_name(struct structure_name variable_name)
{
    //function code that uses the structure variable
}
```

Example:

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

Example: // Input all detail and show all

```
#include <stdio.h>
struct student{
  char name[100];
  int id;
} x[10];
void detail(struct student [], int);
int main()
{
    int i=0, num;
    printf("\n Enter how many details you want: ");
    scanf("%d", &num);
    for(i=0; i<num; i++)
    {
       printf("\n Enter details for %d", i+1);
       printf("\n Name: ");
       scanf gets(x[i].name);
       printf("\n Id: ");
       scanf("%d", &x[i].id);
    }
    detail(x, num);
    return 0;
}
void detail(struct student a[], int n);
{
    int j=0;
```

```
for(j=0 ; j<n , j++)
{
    printf("\n Detail of %d student", i+1);
    printf("\n Name : %s", a[j].name);
    printf("\n Id : %d", a[j].id);
}
}
```

g. Create a structure named "student" that has name, roll and marks as members. Use this structure to read records of 20 students and write those records to a file. Now read back those records from a file and display the information of only those students whose marks is greater than or equal to 40. Assume appropriate data types. [8]

ans.

```
#include<stdio.h>
struct student{
    char name[100];
    int roll;
    float marks;
} stu[20];
int main( )
{
    int i=0
```

10. What do you mean by pointers? Write a program to find the largest array element using pointer. [2+6].

ans. A pointer is a variable that stores the memory address of another variable. Pointers are used to manipulate data and memory allocation. To declare a pointer variable, we need to use * symbol before the variable name. We need to assign the address of a variable to a pointer, use the "&" operator. For eg:

```
int *ptr, num = 10;
ptr = &num;
```

On array, we should assign base address of array to pointer like,

```
int *ptr, num[10];
ptr = num;   or   ptr = &num[0];
```

To access the value of the variable that a pointer is pointing to, we can use the "*" operator,

For eg:

```
int *ptr, num = 10;
int value;
*ptr = &num;
value = *ptr;
```

// To find the largest array element using pointer

```
#include<stdio.h>
int main( )
{
    int a[100], num = 0 , i=0 , gt = 0, *ptr , *p;
    ptr = a;
    p = &num;
    printf("\n How many numbers you want to compare: ");
    scanf ("%d", p);
    for (i=0 ; i< *p ; i++)
    {
        printf("\n Number %d: ", i+1 );
        scanf ("%d", (ptr+i) );
        if (*(ptr+i) > gt )
        {
            gt = *(ptr+i);
        }
    }
    printf("\n The greatest among them is %d", gt );
    return 0;
}
```

11. Write a graphics program to display a square whose sides are 100 pixels. Also draw a circle inside the square such that the circumference of the circle touches all four sides of the square. [8]

ans.

```
#include<stdio.h>
#include<conio.h>          //BGI (Borland Graphics Interface)
#include<graphics.h>
int main( )
{
    int gd=DETECT, gm;
    initgraph(&gd, &gm "C:\\TURBOC3\\BGI");
    rectangle(200, 200, 300, 300);
    circle (250, 250, 50);
    getch( );
    closegraph( );
}
```

12. Write short notes on any Two: [4+4]

a. Algorithm and Flowchart

- An algorithm is a step-by-step procedure for solving a problem in a finite amount of time It is a sequence of instructions that take some input, perform some operations on it, and produce an output, An algorithm must be clear, unambiguous, and well-defined It is generally written in plain English, pseudocode, or a programming language.

b- Flowchart is a graphical representation of an algorithm It is a diagramatic representation of algorithm that shows the steps involved in solving a problem. The Flowchart uses various symbols to represent the different operations performed in algorithm. The Flowchart begins with a start symbol and ends with an end symbol. It also includes symbols for decision-making, input/output, & processing. The Flowchart is used to help understand the algorithm, visualize the solution, & communicate the solutions to others.

The advantages of using algorithm & Flowchart are:

- They help to break down complex problems into smaller, more manageable parts.
- They provide a visual representation of the solution, making it easier to understand.
- They help to identify errors & inefficiencies in the solution.

- They provide a common language for communicating the solution to others.

In conclusion, algorithm & flowchart are important tools for designing & representing a solution to a problem in computer programming. A well-designed algorithm & flowchart can make the problem-solving process more efficient & effective.

b. String handling functions

- String handling functions in programming are used to manipulate strings, which are a sequence of characters. Here are some commonly used string handling function in C programming:

i. strcpy(): // strcpy(destination, source)

This function is used to copy one string to another string. It takes two parameters, the destination string and the source string. The destination string should be large enough to hold the source string.

ii. strcat(): // strcat(destination, source)

This function is used to concatenate two strings. It takes two parameters, the destination string & the source string. The destination string should be large enough to hold both strings.

iii. strlen(): // strlen(string)

This function is used to find the length of a

string. It takes one parameter, the string whose length is to be determined.

4. strcmp( ): // strcmp(string1, string2)

This function is used to compare two strings. It returns 0 if the strings are equal, a positive value if the first string is greater than the second, & a negative value if the first string is less than the second.

5. strlwr( ):

This function is used to converts all uppercase letters in a string to a lowercase letters.

6. strupr( ):

This function is used to converts all lowercase letters in a string to a uppercase letters.

These functions are very useful in string manipulation & can be used in a variety of applications such as text processing, file handling, & network programming. It is important to understand the use of these functions & their limitations in order to write efficient and error-free code.

c. Dynamic Memory Allocation (DMA):-

- DMA is the process of allocating memory at runtime, i.e., when the program is executing; It is a way to allocate memory dynamically to variables when the amount of money required is not known at compile-time.

The dynamic memory allocation functions in C are malloc(), calloc(), realloc(), free().

1. malloc():

The malloc() function is used to dynamically allocate memory in C. It takes a single argument, the number of bytes to allocate, & returns a void pointer to the beginning of allocated memory block.

Syntax:

```
ptr = (cast type *)malloc (n * sizeof (datatype));
```

2. calloc():

The calloc() function is similar to malloc(), but it also initializes the allocated memory block to zero. It takes two arguments, the number of elements allocate and the size of each element in bytes, & returns a void pointer to the allocated memory block.

Syntax:

```
ptr = (cast type *) calloc (n, sizeof (datatype));
```

3. realloc():

The realloc() function is used to resize the previously allocated memory block. It takes two

arguments, a pointer to the previously allocated memory block & the new size of the memory block, & returns a void pointer to the beginning of the resized memory block.

Syntax:

```
ptr = (cast type*) realloc (ptr, n * sizeof(datatype));
```

4. Free( ):

The free function is used to deallocate the previously allocated memory block. It takes a single arguments, a pointer to the previously allocated memory block, and deallocates the memory block.