Theory Final Exam Question

Total Marks 100

Answer Script

Question No. 01

Question:1

a) Define ternary operator. Write a C program to find the maximum of two numbers using ternary operators. **(2+5)**

b) Write down the differences between local and global variables. **(3)**

Answer No. 01

a.

Ternary Operator:

A ternary operator is an operator that takes three arguments as input and returns one output.

variable_name = (condition) ? value_if_true : value_if false;

Code:

```
#include<stdio.h>
int main()
{
   int a, b;
   scanf("%d%d", &a, &b);

   int max = a>b?a:b;

   printf("%d", max);

   return 0;
}
```

b.

The difference between Local variables and Global variables are:

S.No	Local Variables	Global Variables
1	It is declared inside a function.	It is declared outside the function.
2	If it is not initialized, a garbage value is stored.	If it is not initialized zero is stored as default.
3	It is stored on the stack unless	It is stored on a fixed location

	specified.	decided by the compiler.
4.	Local variables are stored in a stack in memory.	Global variables are stored in the data segment of memory.
5.	We can declare various variables with the same name but in other functions.	We cannot declare many variables with the same name.

Question No. 02

Question:2

You want to build a fibonacci series up to n terms. Now solve these problems without using an array and at first write the steps to solve these problems and then write a C program for it.

Note - The Fibonacci sequence is a sequence where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are 0 followed by 1.

Input Format:

You are given a positive integer n.

Output Format:

Print the fibonacci series up to n terms.

See the sample input output for more clarification

Sample input	Sample output
10	0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Answer No. 02

Steps:

1. First take an input how many fibonacci numbers the user should want. Then store it n variables.
2. Then declare three variables a, b and x and store a = 0, b = 1.
3. Then check n value. If n=1 then print a, or if n=2 then print 2 values a and b, or if n value is greater than 2 then go to the next step.
4. First print a and b values. Then make a for loop which works 1 to (n-2) times. Then add a with b and store the sum in variable x. Then print x value. Then, Last store b value in a

and x value in b variable for next iteration. These steps work until the for loop condition goes to false.

Code:

```
#include<stdio.h>
int main()
{
   int n;
   scanf("%d", &n);

   int a=0, b=1;

   if(n==1)
      printf("%d", a);
   else if(n==2)
      printf("%d, %d", a, b);
   else if(n>2)
   {
      printf("%d, %d", a, b);
      int x;

      for(int i=1; i<=(n-2); i++)
      {
         x = a + b;
         printf(", %d", x);
         a = b;
         b = x;
      }
   }

   return 0;
}
```

Question No. 03

Question - 3

Define pointers of C. What are the differences between pass by value and pass by references, Explain it with C program using function. **(1+3+6)**

Answer No. 03

Pointers:

The Pointer in C, is a variable that stores address of another variable. A pointer can also be used to refer to another pointer function.

The differences between pass by value and pass by references are:

S.No	Pass by value	Pass by references
1.	When a method is called, the caller passes a copy of the argument variables to the method resulting in two values in memory.	When a method is called, the method arguments reference the same variable in memory as the caller.
2.	The actual value that is passed as argument is not changed after performing some operation on it.	The actual value that is passed as argument is changed after performing some operation on it.
3.	It creates a copy of that variable into the stack section in memory.	It creates a copy of the reference of that variable into the stack section in memory. It uses a reference to get the value.
4.	It changes the value of that copy, the actual value remains the same.	It changes the value of the actual variable.
5.	There is no modification in the original value.	There is a modification in the original value.

Explanation:

In call by value it creates a copy of that variable into the stack section in memory. When the value is changed, it changes the value of that copy, the actual value remains the same.

But in call by reference, it creates a copy of the reference of that variable into the stack section in memory. If we change the value using the reference it changes the value of the actual variable.

Code:

Pass by value:

```
#include<stdio.h>

void func(int x)
{
   x = 50;
   printf("%d\n", x);
}

int main()
{
   int n;
   scanf("%d", &n);

   func(n);

   printf("%d", n);

   return 0;
}
```

Pass by references:

```
#include<stdio.h>

void func(int *x)
{
   *x = 50;
   printf("%d\n", *x);
}

int main()
{
   int n;
   scanf("%d", &n);

   func(&n);

   printf("%d", n);

   return 0;
}
```

Question No. 04

Question - 4

You are given a string S of small letters.Now sort the string in ascending order using frequency array.Write the steps to solve this problem with proper diagram.Lastly write a C program for it and use function in this program. **(10)**

Note - The given input may contain duplicate characters

Sample input	**Sample output**
adsarbrro	aabdorrrs

Answer No. 04

Steps:

1. First we take string input.
2. Then declare a 26 size frequency array and set 0 in this array every index value.
3. Then create a for loop i=0 to string length times. In this loop every iteration i'th index character subtract by 'a' and subtract value + 1 index value every time increases by 1.
4. Again make a for loop and count how many indexes in the frequency array is equal or greater than 1.
5. Then make a while loop. Declare a variable next = 0. This loop works until the next equal to total number of indexes in the frequency array is equal or greater than 1. Inside this while loop make a for loop i=1 to 26 times. Every time it's check i'th index frequency value is greater than 0 or not. If the condition is true then go inside and store i'th frequency value in max variable, i'th value + 96 in ch variable and index number.
6. Finally print this character max time and set 0 in the frequency array index'th value and next value increase by 1.

Diagram:

Step: 2

freq[0]	freq[1]	freq[2]	freq[3]	freq[4]	freq[5]	freq[6]	freq[7]	freq[8]	freq[9]
0	0	0	0	0	0	0	0	0	0

freq[10]	freq[11]	freq[12]	freq[13]	freq[14]	freq[15]	freq[16]	freq[17]	freq[18]	freq[19]
0	0	0	0	0	0	0	0	0	0

freq[20]	freq[21]	freq[22]	freq[23]	freq[24]	freq[25]
0	0	0	0	0	0

Step: 3

freq[0]	freq[1]	freq[2]	freq[3]	freq[4]	freq[5]	freq[6]	freq[7]	freq[8]	freq[9]
0	2	1	0	1	0	0	0	0	0

freq[10]	freq[11]	freq[12]	freq[13]	freq[14]	freq[15]	freq[16]	freq[17]	freq[18]	freq[19]
0	0	0	0	0	1	0	0	3	1

freq[20]	freq[21]	freq[22]	freq[23]	freq[24]	freq[25]
0	0	0	0	0	0

Step: 6

1st iteration:

freq[0]	freq[1]	freq[2]	freq[3]	freq[4]	freq[5]	freq[6]	freq[7]	freq[8]	freq[9]
0	0	1	0	1	0	0	0	0	0

freq[10]	freq[11]	freq[12]	freq[13]	freq[14]	freq[15]	freq[16]	freq[17]	freq[18]	freq[19]
0	0	0	0	0	1	0	0	3	1

freq[20]	freq[21]	freq[22]	freq[23]	freq[24]	freq[25]
0	0	0	0	0	0

2nd iteration:

freq[0]	freq[1]	freq[2]	freq[3]	freq[4]	freq[5]	freq[6]	freq[7]	freq[8]	freq[9]
0	0	0	0	1	0	0	0	0	0

freq[10]	freq[11]	freq[12]	freq[13]	freq[14]	freq[15]	freq[16]	freq[17]	freq[18]	freq[19]
0	0	0	0	0	1	0	0	3	1

freq[20]	freq[21]	freq[22]	freq[23]	freq[24]	freq[25]
0	0	0	0	0	0

Code:

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

void solve(char s[])
{

   int len = strlen(s);

   int freq[27];

   for(int i=0; i<27; i++)
   {
      freq[i] = 0;
   }

   for(int i=0; i<len; i++)
   {
      freq[(s[i] - 'a') + 1]++;
   }

   int total = 0;
   for(int i=1; i<27; i++)
   {
      if(freq[i]>=1)
      {
         total++;
      }
   }
```

```
    int next=0;
    while(next!=total)
    {
        int max=-1, index=-1;
        char ch;

        for(int i=1; i<=26; i++)
        {
            if(freq[i]>=1)
            {
                max = freq[i];
                ch = i+96;
                index = i;
                break;
            }
        }

        for(int i=1; i<=max; i++)
        {
            printf("%c", ch);
        }
        freq[index] = 0;
        next++;
    }
}


int main()
{
    char s[1000];
    gets(s);

    solve(s);

    return 0;
}
```

Question No. 05

Question - 5

What is the difference between malloc and calloc? Write a c program to take **N** elements in an array. You must use malloc to declare this array and show the output of the array. **(4+6)**

Answer No. 05

The difference between malloc and calloc are:

S.No	malloc	calloc
1.	malloc() function creates a single block of memory of a specific size.	calloc() function assigns multiple blocks of memory to a single variable.
2.	malloc() is faster.	calloc() is slower.
3.	The number of arguments in malloc() is 1.	The number of arguments in calloc() is 2.
4.	malloc() has high time efficiency.	calloc() has low time efficiency.
5.	malloc() indicates memory allocation.	calloc() indicates contiguous allocation.

Code:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int n, i;
   scanf("%d", &n);

   int* ptr;

   ptr = (int*) malloc(n*sizeof(int));

   for(i=0; i<n; i++)
   {
      scanf("%d", (ptr+i));
   }

   for(i=0; i<n; i++)
   {
      printf("%d ", *(ptr+i));
   }
```

```
    free(ptr);

    return 0;
}
```

Question No. 06

Question - 6

What is a function? Types of user-defined functions explain with proper examples. What are the benefits of using user-defined functions? **(2+5+3)**

Answer No. 06

Function:

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function.

There are four types of User-defined functions in C.

Those are:

1. Function with no arguments and no return value.
2. Function with no arguments and a return value.
3. Function with arguments and no return value.
4. Function with arguments and with return value.

Explain with Example:

1. Function with no arguments and no return value:

Functions that have no arguments and no return values.

Code:

```
#include<stdio.h>

void solve()
{
   int a, b;
   scanf("%d%d", &a, &b);
   printf("%d %d", a, b);
}

int main()
{
   solve();
   return 0;
}
```

In the above program, function solve does not take any arguments and has no return values. It takes a and b as inputs from the user and prints them inside the void function.

2. Function with no arguments and a return value:
Functions that have no arguments but have some return values.

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

int solve()
{
   int a, b;
   scanf("%d%d", &a, &b);

   return a+b;
}

int main()
{
   int ans;
   ans = solve();
   printf("%d", ans);
   return 0;
}
```

In the above program, function solve does not take any arguments and has a return value as an integer type. It takes a and b as inputs from the user and returns their sum.

3. Function with arguments and no return value
Functions that have arguments but no return values.

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

void solve(int a, int b)
{
   printf("%d %d", a, b);
}

int main()
{
   int a, b;
   scanf("%d%d", &a, &b);

   solve(a, b);

   return 0;
```

```
}
```

In the above program, function solve does take a and b as arguments and has no return value. The main function takes a and b as inputs from the user and calls the solve function to perform the print operation on the given arguments.

4. Function with arguments and with return value:

Functions that have arguments and some return value.

Code:

```
#include<stdio.h>

int solve(int a, int b)
{
   return a+b;
}

int main()
{
   int a, b;
   scanf("%d%d", &a, &b);

   int ans = solve(a, b);
   printf("%d", ans);
   return 0;
}
```

In the above program, function solve takes two arguments as a and b, and has a return value as an integer type. The main function takes input a and b from the user and calls the solve function to perform a specific operation on the given arguments and returns their sum.

The benefits of using user-defined functions:

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs.
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Question No. 07

Question - 7

You have been given **an NxM** matrix, Now your task is to take two matrix as input and find the sum of this two matrix. **(10)**

Sample Input	**Sample Output**
3 3 1 2 3 4 5 6 7 8 9 1 1 1 3 2 4 2 1 2	2 3 4 7 7 10 9 9 11

Answer No. 07

Code:

```
#include<stdio.h>
int main()
{
   int n, m;
   scanf("%d%d", &n, &m);

   int ara1[n+1][m+1];
   int ara2[n+1][m+1];
   int ara3[n+1][m+1];

   for(int i=0; i<n; i++)
   {
      for(int j=0; j<m; j++)
      {
         scanf("%d", &ara1[i][j]);
      }
   }

   for(int i=0; i<n; i++)
   {
      for(int j=0; j<m; j++)
      {
```

```
            scanf("%d", &ara2[i][j]);
        }
    }


    for(int i=0; i<n; i++)
    {
        for(int j=0; j<m; j++)
        {
            ara3[i][j] = ara1[i][j] + ara2[i][j];
        }
    }

    for(int i=0; i<n; i++)
    {
        for(int j=0; j<m; j++)
        {
            printf("%d ", ara3[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Question No. 08

Question -8

a) What is structure? Explain Why we use it? **(2)**

b) How many ways to access structure members? Explain with example. **(3)**

c) Write a C program to store the information of a student using structure. **(5)**

Sample Input	**Sample Output**
Enter information: Enter name: Vanya Enter roll number: 25 Enter marks: 80	Displaying Information: Name: Vanya Roll number: 25 Marks: 80

Answer No. 08

a.

Structure:

Structures are a way to group several related variables into one place. Each variable in the structure is known as a member of the structure.

A structure is used to represent information about something more complicated than a single number, character, or boolean can do. For example, a student can be defined by his or her name, mobile number, age, id etc.

b.

There are two ways to access structure members

1. (.) or Dot Operator
2. Arrow Operator(->)

Dot Operator:

We can access the member roll and age of Student structure variable s as:
s.roll and s.age .

Code:

```
#include<stdio.h>

struct Student{
   int roll;
   int age;
```

```
};

int main()
{
   struct Student s = {12, 22};

   printf("Roll: %d\nAge: %d\n", s.roll, s.age);

   return 0;
}
```

Arrow Operator:

Structure pointer operator or Arrow operator is used to access members of structure using pointer variable. When we have pointer to a structure variable, then we can access member variable by using pointer variable followed by an arrow operator and then the roll and age of the member variable.

Code:

```
#include<stdio.h>

struct Student{
   int roll;
   int age;
};

int main()
{
   struct Student s = {15, 23};
   struct Student* p;

   p = &s;

   printf("Roll: %d\nAge: %d\n", p->roll, p->age);

   return 0;
}
```

c.

```
#include<stdio.h>

struct Student{
   char name[50];
   int number;
   float marks;
```

```
};

void display(struct Student s)
{
    printf("Displaying Information:\n");
    printf("Name: %s\n", s.name);
    printf("Roll Number: %d\n", s.number);
    printf("Marks: %.2f\n", s.marks);
}

int main()
{
    struct Student s;

    printf("Enter information:\n");
    printf("Enter name: ");
    scanf("%s", &s.name);
    printf("Enter roll number: ");
    scanf("%d", &s.number);
    printf("Enter marks: ");
    scanf("%f", &s.marks);

    display(s);

    return 0;
}
```

Question No. 09

Question -9

a) Explain 5 types of Errors in C with Example (You have to write program) **(10)**

Answer No. 09

There are 5 types of error in C:

1. Syntax error
2. Run-time error
3. Linker error
4. Logical error
5. Semantic error

Syntax error:

```
int main()
{
   int a  //here semi colon ; missed
}
```

In C programming we know every line ends with a semicolon. But here int a this line doesn't finish with a semicolon. So, this is Syntax error.

Run-time error:

```
int main()
{
   int a = 10;
   int b = a/0;  // Here number divisible zero error occurs
}
```

We know any number couldn't be divided by 0. But here a is divisible by 0. It's not possible, that's why this is a Run-time error.

Linker error:

```
void Main()     //Here Main() method used instead of main() method
{
}
```

We know in C programming every code starts in the main function. We can declare this function like void main(){} or int main(){}. main is a keyword, so we must write main in all

small letters. But here main declared by Main. Where the first letter M is capital. So, this is a linker error.

Logical error:

```
void main()
{
   printf("%d",sum(10,20));
}
int sum(int a, int b)
{
   return a*b;            //expectation is sum but we are multiplying the numbers
}
```

In this code we want the sum of the two numbers. But here the sum function returns multiplication of two numbers. That's why this is a Logical error.

Semantic error:

```
void main()
{
   int x, y, z;
   x + y = z; //semantic error
}
```

In C programming we can store any value in any variable. But it has fixed rules. The result of a+b, a-b, a*b, a/b this operation we can store any variable. Like x = a+b.
But here this code z variables value store x + y.
x, y they both are variables and + is an operator. we can't store any value in x+y.
So, it's semantic error.

Question No. 10

Question - 10

(Think out of the Box)

(5x2 = 10)

a) Why is C called the Mother of all Languages?

b) What is source code & **.obj** file and **.exe** file?

c) When you run a C program How many files are created on your device? What are they?

d) What is the base condition in the recursion function? Describe a real-life example of recursion.

e) Why should we declare large data type first in structure?

Answer No. 10

a.

C is one of the most widely used programming languages of all time, and C compilers are available for the majority of available computer architectures and operating systems. The C language has formed the basis for many languages including C++, Java, JavaScript, Go, Rust, Limbo, LPC, C#, PHP, Python, Perl, Verilog, and C-shell.
That's why C is called the mother of all languages.

b.

Source Code:

Source code is a group of instructions a programmer writes using computer programming languages.

.obj file:

An object file is a computer file containing object code, that is, machine code output of an assembler or compiler. The object code is usually relocatable, and not usually directly executable.

.exe file:

An executable file is a type of computer file that runs a program when it is opened. This means it executes code or a series of instructions contained in the file.

c.

When i run a C program 3 files are created on my device.

These are:

1. .c
2. .obj
3. .exe

d.

The condition that is applied in any recursive function is known as a base condition. A base condition is a must in every recursive function otherwise it will continue to execute like an infinite loop.

Real life Example:

I want to count how many pages remain in my offset paper bundle. So, I divide the remaining paper into two parts. 1 part I took and another part I gave to my friend. My part I count how much paper I have and another part my friend counts how much paper he has in his part. and finally I added my count number with my friend's count number. Then I get the total number of pages I have in my offset paper bundle.

e.

For memory optimization we declare large data type first in structure.