C Program (Short Questions Ans)

1.Explain the basic structure of a C program with an example.

Ans:- A C program consists of several parts, and here is a simple example:

```
#include <stdio.h>
// Function prototype
void greet();
// Main function
int main()
{
   // Statements in the main function
   printf("Hello, World!\n");
   // Call another function
   greet();
   // Return statement
   return 0;
}
// Definition of the greet function
void greet()
{
   // Statements in the greet function
   printf("Greetings from another function!\n");
}
```

Now, let's break down the basic structure of the C program:

1. Header Files:
   - #include <stdio.h>: This line includes the standard input-output library, which is necessary for functions like printf and scanf.
2. Function Prototype:
   - void greet();: This line declares a function prototype for the function greet(). It tells the compiler that there is a function named greet, which will be defined later in the program.
3. Main Function:
   - int main() { /*...*/ }: The main function is the starting point of every C program. Execution begins from here. In this example, it contains statements to print "Hello, World!" using printf, calls the greet function, and returns 0 to indicate successful program execution.
4. Function Definition:
   - void greet() { /*...*/ }: This is the definition of the greet function. It is called from the main function. In this example, it contains statements to print a greeting message.
5. Comments:

- //: These are single-line comments. They are used to provide explanations or annotate the code.

6. Statements:
   - Inside both the main and greet functions, you have statements that perform specific tasks. In this example, the printf function is used to print messages to the console.
7. Return Statement:
   - return 0;: The main function typically ends with a return statement. A return value of 0 conventionally indicates successful program execution, while a non-zero value indicates an error.

This is a simple example, but it covers the essential components of a C program. You have declarations, functions, statements, and comments working together to perform a simple task.

2.What is an assignment statement? Give the general form of an assignment statement?

Ans:- An assignment statement is a fundamental concept in programming languages, including C. It is used to assign a value to a variable. The general form of an assignment statement is as follows:
variable = expression;
Here, **variable** is the name of the variable that you want to assign a value to, and **expression** is the value or the result of an operation that you want to assign to the variable.
Here's an example to illustrate the use of an assignment statement:

```
#include <stdio.h>
int main()
{
  // Declare a variable of type int
  int number;
  // Assign a value to the variable using an assignment statement
  number = 10;
  // Print the value of the variable
  printf("The value of 'number' is: %d\n", number);
  // Update the variable with a new value
  number = number + 5;
  // Print the updated value of the variable
  printf("After adding 5, the value of 'number' is: %d\n", number);
  return 0;
}
```

3.Write a C program to find the largest of three numbers using ternary operator.

Ans:-

Source code:-

```
#include <stdio.h>
int main()
{
   int num1, num2, num3, largest;
   // Input three numbers from the user
   printf("Enter three numbers: ");
   scanf("%d %d %d", &num1, &num2, &num3);
   // Using ternary operator to find the largest number
   largest = (num1 > num2) ? ((num1 > num3) ? num1 : num3) : ((num2 > num3) ?
   num2 : num3);
   // Displaying the result
   printf("The largest number is: %d\n", largest);
   return 0;
}
```

Output:-

```
Enter three numbers: 35 67 89
The largest number is: 89
```

4. Write a "C‟ program to demonstrate the use of unconditional goto statement.

Ans:- The use of goto is generally discouraged in modern programming due to its potential to create unreadable and hard-to-maintain code. However, for the sake of demonstration, here's a simple C program that uses an unconditional goto statement:

```
#include <stdio.h>
 int main()
{
  int i = 1;
  // Loop using goto
  start:
  if (i <= 10)
  {
     printf("%d ", i);
     i++;
     goto start;  // Unconditional goto statement
  }
  printf("\n");
  return 0;
}
```

In this example, the program uses a goto statement to create a loop that prints numbers from 1 to 10. The start: label is defined before the loop, and the goto start; statement is used to jump back to the start label until the condition i <= 10 is no longer true.

It's important to note that using goto in this way is considered bad practice, and there are usually better alternatives such as using loops (**for, while, or do-while**) for better code readability and maintainability.

5. Write a C program to find the sum and average of n integer numbers.

Ans:-

Source code:-

```
#include <stdio.h>
int main()
{
  int n, i, num;
  float sum = 0.0, average;
  // Input the number of elements
  printf("Enter the number of elements: ");
  scanf("%d", &n);
  // Input the numbers and calculate the sum
  for (i = 1; i <= n; ++i)
  {
     printf("Enter number %d: ", i);
     scanf("%d", &num);
     sum += num;
  }
  // Calculate the average
  average = sum / n;
  // Display the sum and average
  printf("Sum = %.2f\n", sum);
  printf("Average = %.2f\n", average);
  return 0;
}
```

Output:-

```
Enter the number of elements: 4
Enter number 1: 5
Enter number 2: 6
Enter number 3: 3
Enter number 4: 7
Sum = 21.00
Average = 5.25
```

6.Explain with syntax and example ,the different string manipulation library functions with example.

Ans:- String manipulation in C is often performed using standard library functions from the <string.h> header. Here are some commonly used string manipulation functions along with explanations and examples:

1.strlen() - String Length:

- Syntax: **size_t strlen(const char *str);**
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
  char str[] = "Hello, World!";
  size_t length = strlen(str);
  printf("Length of the string: %zu\n", length);
  return 0;
 }
```

Output:- **Length of the string: 13**

2.strcpy() - String Copy:

- Syntax: char *strcpy(char *dest, const char *src);
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
  char source[] = "Hello";
  char destination[20];
  strcpy(destination, source);
  printf("Copied string: %s\n", destination);
  return 0;
}
```

Output:- Copied string: Hello

3.strcat() - String Concatenation:

- Syntax: char *strcat(char *dest, const char *src);
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
```

```
{
  char str1[20] = "Hello, ";
  char str2[] = "World!";
  strcat(str1, str2);
  printf("Concatenated string: %s\n", str1);
  return 0;
}
```

Output: Concatenated string: Hello, World!

4.strcmp() - String Comparison:

- Syntax: `int strcmp(const char *str1, const char *str2);`
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
  char str1[] = "apple";
  char str2[] = "orange";
  int result = strcmp(str1, str2);
  printf("Comparison result: %d\n", result);
  return 0;
 }
```

Output: Comparison result: -14 (negative indicates str1 is less than str2)

5.strncpy() - Limited String Copy:

- Syntax: `char *strncpy(char *dest, const char *src, size_t n);`
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
  char source[] = "Hello";
  char destination[5];
  strncpy(destination, source, 4);
  destination[4] = '\0'; // Ensure null-termination
  printf("Copied string: %s\n", destination);
  return 0;
}
```

Output:- Copied string: Hell

7. What are the three components of defining a user defined functions in "C"?

Ans:- In C, when defining a user-defined function, there are three main components:

1. Return Type:
   - The return type specifies the type of the value that the function will return to the calling code. If the function does not return any value, the return type is declared as `void`.
   - Example: `int`, `float`, `double`, `void`, etc.
2. Function Name:
   - The function name is an identifier that uniquely identifies the function. It is used by the calling code to invoke the function.
   - Example: `calculateSum`, `printMessage`, etc.
3. Parameter List:
   - The parameter list contains the variables or values that the function expects to receive from the calling code. These are the input values used by the function during its execution.
   - Example: `(int num1, int num2)` - here, `num1` and `num2` are parameters.

Here is a simple example of a user-defined function in C, illustrating these three components:

Source code:-

```
#include <stdio.h>
// Function declaration
int addNumbers(int num1, int num2);
int main()
{
   int a = 5, b = 7, sum;
   // Function call
   sum = addNumbers(a, b);
   printf("Sum: %d\n", sum);
   return 0;
}
// Function definition
int addNumbers(int num1, int num2)
{
   return num1 + num2;
}
```

Output:-

```
Sum: 12
```

8.Explain the difference between array and structure.

Ans:-

Feature	Array	Structure
Definition	An array is a collection of elements of the same data type stored in contiguous memory locations.	A structure is a user-defined data type that allows bundling of variables of different data types under a single name.
Data Type	All elements in an array must be of the same data type.	Elements in a structure can be of different data types. Each element is called a member or field.
Size	The size of an array is fixed and determined at compile-time.	The size of a structure is the sum of the sizes of its members. It can vary based on the types and number of members.
Accessing Elements	Elements in an array are accessed using an index.	Members in a structure are accessed using dot notation (.) with the member name.
Memory Allocation	Contiguous block of memory is allocated for all elements in an array.	Members in a structure are allocated separate memory locations.
Usage	Suitable when dealing with a collection of similar elements.	Suitable when dealing with a group of related but different elements.
Initialization	Arrays can be initialized at the time of declaration or later using a loop or individual assignments.	Members of a structure can be initialized individually using curly braces {} during declaration or later using assignment statements.
Example	c int numbers[5] = {1, 2, 3, 4, 5};	c struct Point { int x; int y; }; struct Point p1 = {10, 20};
Example (Dynamic)	Arrays can be dynamically allocated using functions like malloc in C.	Structures can also be dynamically allocated, but each member must be individually allocated.
Passing to Functions	When passing an array to a function, a pointer to the first element is commonly used.	When passing a structure to a function, it can be passed by value or by reference (using pointers).
Usage in Mathematics	Suitable for mathematical operations involving homogeneous data.	Useful for representing complex entities with different attributes, like a point in 2D space.

9.Explain with example, the various constants available in "C‟ language.

Ans:- Here's an explanation with examples for various constants:

1.Integer Constants:

- Integer constants are whole numbers without any fractional part.
- Examples:

```
10      // Decimal constant
0x1A    // Hexadecimal constant (starts with 0x)
077     // Octal constant (starts with 0)
```

2.Floating-point Constants:

- Floating-point constants represent real numbers with a fractional part.
- Examples:

```
3.14    // Standard decimal representation
2.0e3   // Exponential notation (2.0 * 10^3)
```

3.Character Constants:

- Character constants represent individual characters enclosed in single quotes.
- Examples:

```
'A'     // Character constant
'\n'    // Escape sequence for a newline character
```

4.String Constants:

- String constants are sequences of characters enclosed in double quotes.
- Example:

```
"Hello, World!"    // String constant
```

5.Enumeration Constants:

- Enumeration constants are user-defined constants created using the `enum` keyword.
- Example:

```
enum Color
{
  RED,
  GREEN,
  BLUE
};
```

Here, `RED`, `GREEN`, and `BLUE` are enumeration constants with integer values 0, 1, and 2, respectively.

6.Symbolic Constants:

- Symbolic constants are defined using the `#define` preprocessor directive. They are often used to create named constants for improved code readability.
- Example:

```
#define PI 3.14159265
```

Here, **PI** is a symbolic constant representing the value of pi.

These examples illustrate the various types of constants in the C programming language.

10. Show how break and continue statements are used in a C program, with example.

Ans:-

Example with break statement:

The break statement is used to exit a loop prematurely, typically based on some condition. In the example below, a for loop is used to iterate from 1 to 10, but the loop is set to break when the loop variable i reaches 5.

```
#include <stdio.h>
int main()
{
   for (int i = 1; i <= 10; i++)
   {
      if (i == 5)
     {
         printf("Breaking out of the loop at i = %d\n", i);
         break;
      }
      printf("%d ", i);
    }
   return 0;
 }
```

Output:-

1 2 3 4 Breaking out of the loop at i = 5

In this example, the loop breaks when i equals 5, and the program jumps out of the loop.

Example with continue statement:

The continue statement is used to skip the rest of the loop body and move to the next iteration of the loop. In the following example, a for loop is used to iterate from 1 to 5, but the loop is set to continue when the loop variable i equals 3.

```
#include <stdio.h>
int main()
{
   for (int i = 1; i <= 5; i++)
   {
      if (i == 3)
     {
         printf("Skipping i = %d\n", i);
         continue;
      }
```

```
        printf("%d ", i);
    }
    return 0;
}
```

Output:-

```
1 2 Skipping i = 3
4 5
```

In this example, when i equals 3, the continue statement is executed, skipping the printf statement and moving to the next iteration of the loop.

11.Write a C program to find the largest element in an array.

Ans:-

Source code:-

```
#include <stdio.h>
int main()
{
    // Define an array
    int arr[] = {10, 5, 7, 2, 8, 15};
    // Find the number of elements in the array
    int size = sizeof(arr) / sizeof(arr[0]);
    // Assume the first element is the largest
    int largest = arr[0];
    // Iterate through the array to find the largest element
    for (int i = 1; i < size; i++)
   {
        if (arr[i] > largest)
       {
            largest = arr[i];
        }
    }
    // Display the result
    printf("The largest element in the array is: %d\n", largest);
    return 0;
}
```

Output:-

```
The largest element in the array is: 15
```

12.Differentiate between call by value and call by reference with examples.

Ans:-

Feature	Call by Value	Call by Reference
Naming Convention	The function is invoked by passing the parameter's value in call by value, which is why it has this name.	The function is invoked by passing the parameter's reference (i.e, the location of variables) in call by reference, which is why it has this name.
Type of passing	Call by value passes a copy of variable.	Call by reference passes the variable itself.
Parameter Passing Mechanism	Values of actual parameters are passed to the function.	Addresses (references) of actual parameters are passed to the function.
Changes to Parameters	Changes made to the parameters inside the function do not affect the actual parameters outside the function.	Changes made to the parameters inside the function directly affect the actual parameters outside the function.
Memory Usage	Requires additional memory for storing copies of values passed.	Requires less memory as it passes the address/reference rather than the actual data.
Syntax in Function Call	function_name(parameter1, parameter2, ...);	function_name(&parameter1, &parameter2, ...);

Example	#include <stdio.h>
	void swapx (int x,int y);
	int main ()
	{
	int a=10,b=20;
	swapx (a,b);
	printf ("In caller a =%d and b =%d",a,b);
	return 0;
	}
	void swapx (int x,int y)
	{
	int t;
	t=x;
	x=y;
	y=t;
	printf ("Inside function x =%d and y =%d\n",x,y);
	}
		#include <stdio.h>
		void swapx (int *,int *);
		int main ()
		{
		int a=10,b=20;
		swapx (&a,&b);
		printf ("In caller a =%d and b =%d",a,b);
		return 0;
		}
		void swapx (int *x,int *y)
		{
		int t;
		t=*x;
		*x=*y;
		*y=t;
		printf ("Inside function x =%d and y =%d\n",*x,*y);
		}

Feature	Call by Value	Call by Reference
Output	Inside function x=20 and y=10 In caller a=10 and b=20	Inside function x=20 and y=10 In caller a=20 and b=10

13. What are actual parameters and formal parameters? Illustrate with example.

Ans:- In C programming, actual parameters and formal parameters are terms related to function parameters.

Formal Parameters:

- Formal parameters are the parameters declared in the function prototype or definition.
- They act as placeholders for the values that will be passed to the function when it is called.
- Formal parameters are also known as "parameters" or "function parameters" in general.
- Example:

Actual Parameters:

- Actual parameters are the values or expressions passed to a function during its call.
- They are the real data that the function uses while executing its code.
- Actual parameters are also known as "arguments" or "function arguments."

To summarize, formal parameters are the variables declared in a function's parameter list, and actual parameters are the values or expressions passed to the function when it is called. The actual parameters provide the actual data that the function will operate on.

Here's an example in C to illustrate this concept:

```
#include <stdio.h>
// Function definition with formal parameters
void addNumbers(int a, int b)
 {
   int sum = a + b;
   printf("Sum of %d and %d is: %d\n", a, b, sum);
}
int main()
{
   int x = 5, y = 3;
   // Function call with actual parameters
   addNumbers(x, y);
   return 0;
 }
```

Output:-

Sum of 5 and 3 is: 8

14. Explain with example how to create a structure using "typedef".

Ans:- In C, the typedef keyword is used to create an alias or a new name for an existing data type. This is often used to simplify complex type declarations, and it can also be employed when defining structures. Here's an example of creating a structure using typedef in C:

Source code:-

```
#include <stdio.h>
// Define a structure without typedef
struct Point
{
   int x;
   int y;
};
// Define a structure with typedef
typedef struct
{
   int x;
   int y;
} Point2D;
int main()
{
   // Using the structure without typedef
   struct Point p1;
   p1.x = 10;
   p1.y = 20;
   printf("Point without typedef: (%d, %d)\n", p1.x, p1.y);
   // Using the structure with typedef
   Point2D p2;
   p2.x = 30;
   p2.y = 40;
   printf("Point with typedef: (%d, %d)\n", p2.x, p2.y);
   return 0;
 }
```

Output:-

```
Point without typedef: (10, 20)
Point with typedef: (30, 40)
```

15. Write short notes on fseek().

Ans:- The fseek() function in C is used to set the file position indicator to a specified position within a file. It allows you to move the position indicator to a specific byte offset in the file, facilitating random access to different parts of the file. This function is part of the Standard Input/Output Library (stdio.h) in C.

Here is the basic syntax of the fseek() function:

```
int fseek(FILE *stream, long offset, int whence);
```

- stream: A pointer to the FILE object representing the file.
- offset: The number of bytes to move the file position indicator. It can be positive or negative.
- whence: It specifies the reference point for the offset. It can take one of the following values:
  - SEEK_SET: Beginning of the file.
  - SEEK_CUR: Current position indicator.
  - SEEK_END: End of the file.

The fseek() function returns 0 on success and a non-zero value on failure.

Here's a simple example demonstrating the use of fseek():

```
#include <stdio.h>

int main()

{

    FILE *file = fopen("example.txt", "r");
    if (file == NULL)
    {
        perror("Error opening file");
        return 1;
    }
    // Move the file position indicator to the 10th byte from the beginning of the file
    if (fseek(file, 10, SEEK_SET) == 0)
    {
        // Read and print the character at the current position
        int ch = fgetc(file);
        printf("Character at position 10: %c\n", ch);
    }
    else
    {
        perror("Error using fseek");
    }
    fclose(file);
    return 0;
}
```

In this example, the program opens a file named "example.txt" and uses fseek() to move the file position indicator to the 10th byte from the beginning of the file (SEEK_SET). It then reads and prints the character at that position. Keep in mind that error handling is essential when working with file operations, and the program checks for errors after each operation.

16. Write a C program that reads from the user an arithmetic operator and two operands, perform the corresponding arithmetic operation on the operands using switch statement.

Ans:-

Source code:-

```
#include <stdio.h>
int main()
{
   // Declare variables to store the operands and the operator
   double num1, num2;
   char op;
   // Prompt the user to enter the operands and the operator
   printf("Enter the first operand: ");
   scanf("%lf", &num1);
   printf("Enter the second operand: ");
   scanf("%lf", &num2);
   printf("Enter the arithmetic operator (+, -, *, /): ");
   scanf(" %c", &op); // Note the space before %c to skip whitespace
   // Use a switch statement to perform the corresponding operation
   switch (op)
   {
   case '+':
      printf("%lf + %lf = %lf\n", num1, num2, num1 + num2);
      break;
   case '-':
      printf("%lf - %lf = %lf\n", num1, num2, num1 - num2);
      break;
   case '*':
      printf("%lf * %lf = %lf\n", num1, num2, num1 * num2);
      break;
   case '/':
      // Check for division by zero
      if (num2 == 0)
      {
         printf("Error: Cannot divide by zero\n");
      }
      else
```

```
        {
            printf("%lf / %lf = %lf\n", num1, num2, num1 / num2);
        }
        break;
    default:
        // Handle invalid operator
        printf("Error: Invalid operator\n");
        break;
    }
    return 0;
}
```

Output:-

```
Enter the first operand: 6
Enter the second operand: 4
Enter the arithmetic operator(+,-,*,/): +
6.000000+ 4.000000 = 10.000000
```

17. What is dangling else problem? Explain how to handle its in C programming.

Ans:- The "dangling else" problem is a situation that can occur in programming languages with conditional statements, where an ambiguous association between an "if" statement and an "else" statement leads to confusion regarding which "if" statement the "else" belongs to. This ambiguity arises when there are nested "if-else" statements, and the programmer hasn't made it clear which "if" corresponds to which "else."

Consider the following example:

```
if (condition1)
    if (condition2)
        statement1;
else
statement2;
```

In this case, it is not clear whether the "else" belongs to the first "if" or the second "if." This ambiguity can lead to unexpected behavior, as the compiler might interpret it differently than the programmer intended.To handle the dangling else problem in C programming, it is a good practice to use braces ({})to explicitly define the scope of the "if" and "else" statements. By doing this, you make it clear which "if" statement the "else" belongs to. Here's the corrected version:

```
if (condition1)
{
    if (condition2)
    {
        statement1;
    }
```

```
        }
        else
        {
           statement2;
}
```

By adding braces, you explicitly define the scope of each "if" and "else" statement, making the code more readable and removing any ambiguity related to the dangling else problem. It's generally a good practice to use braces even when they are not strictly necessary, as it improves code readability and reduces the likelihood of introducing bugs due to misinterpretation of the code structure.

18. Write a program to replace each constant in a string with the text one except letter „z" „Z" and „a" „A" . Thus the string "Programming in C is fun" should be modified as "Qsphsannjoh jo D jt gvo".

Ans:- Below is a simple C program that replaces each constant in a string with the text "one" except for the letters 'z', 'Z', 'a', and 'A'. The program uses a loop to iterate through each character in the input string and performs the replacement based on the specified conditions:

```
#include <stdio.h>
void replaceConstants(char *str)
{
  while (*str != '\0')
  {
     if ((*str >= 'b' && *str <= 'y') || (*str >= 'B' && *str <= 'Y'))
    {
        // Replace constant with "one"
        printf("one");
     }
     else
     {
        // Preserve 'z', 'Z', 'a', 'A'
        putchar(*str);
     }
     str++;
   }
}
int main()
{
  // Example string
  char inputString[] = "Programming in C is fun";
  // Replace constants
  replaceConstants(inputString);
```

```
return 0;
}
```

This program defines a function replaceConstants that takes a string as input, iterates through each character, and prints "one" for constants (excluding 'z', 'Z', 'a', 'A'). The main function demonstrates the usage by passing the example string "Programming in C is fun."

19. What is Recursion? Write a C program to compute polynomial co-efficient nCr using recursion.

Ans:- Recursion is a programming concept in which a function calls itself either directly or indirectly in order to solve a problem. Recursive functions have two main components: a base case and a recursive case. The base case provides a termination condition, and the recursive case breaks the problem down into smaller subproblems.

Now, let's write a C program to compute the binomial coefficient (nCr) using recursion. The binomial coefficient is defined as the number of ways to choose 'r' elements from a set of 'n' distinct elements.

Source code:-

```
#include <stdio.h>
// Function to compute nCr using recursion
int nCr(int n, int r)
{
  // Base case: nC0 and nCn are always 1
  if (r == 0 || r == n)
  {
     return 1;
  }
  else
  {
     // Recursive case: nCr = (n-1)C(r-1) + (n-1)Cr
     return nCr(n - 1, r - 1) + nCr(n - 1, r);
   }
}
int main()
{
  int n, r;
  // Input values for n and r
  printf("Enter the values for n and r (separated by space): ");
  scanf("%d %d", &n, &r);
  // Check if the input values are valid
  if (n < 0 || r < 0 || r > n)
  {
     printf("Invalid input. Make sure n is non-negative and r is between 0 and n.\n");
   }
```

```
  else
   {
     // Compute and print nCr using recursion
     printf("The value of %dC%d is: %d\n", n, r, nCr(n, r));
   }
  return 0;
 }
```

Output:-

```
Enter the values for n and r (separated by space): 2  1
The value of 2c1 is : 2
```

20. What is a macro ?Write a macro to determine whether the given number is odd or even.

Ans:- In C programming, a macro is a fragment of code that has been given a name. It is defined using the #define preprocessor directive. Macros are often used to create reusable pieces of code or to define constants. They are processed by the preprocessor before the actual compilation of the code.

Here's an example of a macro to determine whether a given number is odd or even:

```
#include <stdio.h>
// Macro to check if a number is even
#define IS_EVEN(num) ((num) % 2 == 0)
int main()
{
   int number;
   // Input the number
   printf("Enter a number: ");
   scanf("%d", &number);
   // Check if the number is even using the macro
   if (IS_EVEN(number))
   {
     printf("%d is even.\n", number);
    }
    else
    {
     printf("%d is odd.\n", number);
    }
  return 0;
 }
```

Output:-

```
Enter no: 4
4 is even.
```