Chapter 2:Basics of C programming

2.1 Different approaches in programming: Procedural approach, Object Oriented approach, Event Driven approach

Procedural Approach: The procedural approach is a programming paradigm that focuses on writing a sequence of instructions or procedures to solve a problem. It involves breaking down a problem into smaller, manageable tasks and then designing functions or procedures to accomplish those tasks. In this approach, the emphasis is on the step-by-step execution of instructions, and data is treated as separate from the procedures.

Object-Oriented Approach: The object-oriented approach is a programming paradigm that organizes software design around objects, which are instances of classes. It aims to represent real-world entities as objects that have properties (attributes) and behaviors (methods). Objects communicate with each other by sending messages, and the focus is on data encapsulation, inheritance, and polymorphism. This approach provides reusability, modularity, and extensibility in programming.

Event-Driven Approach: The event-driven approach is a programming paradigm where the flow of the program is determined by events. Events can be user actions, such as mouse clicks or keyboard inputs, or system-generated events, such as timer expirations or network events. In this approach, the program responds to events by executing event handlers or callbacks, which are functions or methods specifically designed to handle specific events. The event-driven approach is commonly used in graphical user interfaces (GUIs) and interactive applications.
Differences between the approaches:

1. Approach to Problem Solving:

- Procedural: Focuses on breaking down a problem into a sequence of steps and designing procedures to solve them.
- Object-Oriented: Focuses on representing problem entities as objects with properties and behaviors and designing classes to encapsulate data and methods.

- Event-Driven: Focuses on designing event handlers or callbacks to respond to specific events in an application.

2. Data Handling:

- Procedural: Treats data and procedures as separate entities.
- Object-Oriented: Emphasizes encapsulating data and methods within objects.
- Event-Driven: Handles data and procedures based on events and their associated event handlers.

3. Code Reusability:

- Procedural: Code reusability is achieved through functions or procedures.
- Object-Oriented: Code reusability is achieved through inheritance and polymorphism.
- Event-Driven: Code reusability is achieved through reusable event handlers or callbacks.

4. Modularity:

- Procedural: Focuses on dividing the program into smaller procedures or functions.
- Object-Oriented: Focuses on dividing the program into classes and objects.
- Event-Driven: Focuses on dividing the program into event handlers or callbacks.

2.2 Structure of C: Header and body, Use of comments, Compilation of a program.

Structure of C: In C programming language, a program is structured into various components, including headers, body, and comments.

- Header: The header section of a C program includes necessary preprocessor directives and function declarations. It is typically placed at the beginning of the program and provides information to the compiler about the libraries or modules to be used.
- Body: The body section of a C program contains the main logic and implementation of the program. It consists of functions, variables, control structures, and any other necessary code.

- Comments: Comments are used to add explanatory or descriptive text within the program. They are ignored by the compiler and do not affect the execution of the program. Comments are helpful for documentation and readability purposes.

Example Program:

```
#include <stdio.h> // Header file
int main() { // This is a simple C program to print "Hello, World!" on the console
printf("Hello, World!");    // Print statement

return 0;

}
```

Explanation:

- The program begins with the inclusion of the "stdio.h" header file, which provides the necessary functions for input and output operations.
- The main function is the entry point of the program.
- Within the main function, there is a single statement using the printf function to print the string "Hello, World!" on the console.
- The return statement is used to exit the main function and return the value 0 to indicate successful execution to the operating system.
- Comments are added to provide information about the purpose and functionality of the code.

Compilation of a Program: To compile a C program, you need a C compiler. The compilation process involves translating the source code into machine-readable instructions.

For example, using the GCC compiler, you can compile a C program named "program.c" by running the following command in the terminal:

```
gcc program.c -o program
```

This command will generate an executable file named "program" if there are no compilation errors. You can then run the program by executing the generated executable file.

2.3Data Concepts: Variables, Constants, data types like: int, float char, double and void. Qualifiers: short and long size qualifiers, signed and unsigned qualifiers. Declaring variables, Scope of the variables according to block, Hierarchy of data types.

Data Concepts:

Variables: Variables are named memory locations used to store values in a program. They can hold different types of data and their values can be changed during program execution.
Constants: Constants are fixed values that cannot be changed during program execution. They are used to represent fixed values such as numbers, characters, or string literals.
Data Types:

- int: Used to store integer numbers.
- float: Used to store floating-point numbers with single precision.
- char: Used to store single characters.
- double: Used to store floating-point numbers with double precision.
- void: Used to indicate the absence of a data type or to define functions that do not return a value.

Qualifiers:

- short and long size qualifiers: These qualifiers modify the storage size of integer data types. "short" reduces the storage size, while "long" increases it.
- signed and unsigned qualifiers: These qualifiers determine whether a numeric data type can represent both positive and negative values ("signed") or only non-negative values ("unsigned").

Declaring Variables: Variables are declared by specifying the data type followed by the variable name. For example:

```
int age; // Declares an integer variable named "age" float pi = 3.14; // Declares and
initializes a float variable named "pi" char letter = 'A'; // Declares and initializes a char
variable named "letter"
```

Scope of Variables: The scope of a variable refers to the part of the program where the variable is visible and can be accessed. Variables can have local scope (limited to a specific block or function) or global scope (accessible throughout the program).
Hierarchy of Data Types: In C, the hierarchy of data types is as follows (from smallest to largest size):

- char
- short int
- int
- long int
- float
- double

Example Program:

```
#include <stdio.h>
int main() { int num1 = 10; // Declaring and initializing an integer variable float num2 =
3.14; // Declaring and initializing a float variable char letter = 'A'; // Declaring and
initializing a char variable
printf("Number 1: %d\n", num1);
printf("Number 2: %.2f\n", num2);
printf("Letter: %c\n", letter);

return 0;

}
```

Explanation:

- In this program, we declare and initialize variables of different data types: int, float, and char.
- We then use the printf function to display the values of these variables on the console.
- The format specifiers (%d, %.2f, %c) are used to specify the data type of the variables being printed.
- The output will display the values of the variables: Number 1: 10, Number 2: 3.14, Letter: A.

2.4 Operators in C:Logical , Arithmetic, Bitwise, Relational, Assignment

Operators in C:

1. Arithmetic Operators:

- Arithmetic operators are used to perform mathematical calculations.
- Examples: + (addition), - (subtraction), * (multiplication), / (division), % (modulus).

Example: int a = 10; int b = 4; int sum = a + b; // sum = 14 int product = a * b; // product = 40 int quotient = a / b; // quotient = 2 int remainder = a % b; // remainder = 2

2. Relational Operators:

- Relational operators are used to compare values.
- Examples: == (equal to), != (not equal to), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to).

Example: int a = 5; int b = 7; int c = 5;
bool isEqual = (a == b); // isEqual = false bool isNotEqual = (a != b); // isNotEqual = true bool isGreater = (a > b); // isGreater = false bool isLess = (a < b); // isLess = true bool isGreaterOrEqual = (a >= c); // isGreaterOrEqual = true bool isLessOrEqual = (a <= c); // isLessOrEqual = true

3. Logical Operators:

- Logical operators are used to combine multiple conditions or evaluate the truth value of expressions.
- Examples: && (logical AND), || (logical OR), ! (logical NOT).

Example: int age = 25; bool isStudent = false;

bool isTeenager = (age >= 13 && age <= 19); // isTeenager = false bool isAdult = (age >= 18 || isStudent); // isAdult = true bool isNotStudent = !isStudent; // isNotStudent = true

4. Bitwise Operators:
- Bitwise operators perform operations on individual bits of binary numbers.
- Examples: & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), >> (right shift).

Example: int a = 5; // binary: 0101 int b = 3; // binary: 0011
int bitwiseAnd = a & b; // bitwiseAnd = 1 (binary: 0001) int bitwiseOr = a | b; // bitwiseOr = 7 (binary: 0111) int bitwiseXor = a ^ b; // bitwiseXor = 6 (binary: 0110) int bitwiseNot = ~a; // bitwiseNot = -6 (binary: 1010) int leftShift = a << 2; // leftShift = 20 (binary: 10100) int rightShift = a >> 1; // rightShift = 2 (binary: 0010)

5. Assignment Operators:
- Assignment operators are used to assign values to variables.
- Examples: = (simple assignment), += (addition assignment), -= (subtraction assignment), *= (multiplication assignment), /= (division assignment), %= (modulus assignment).

Example: int num = 10;
num += 5; // num = 15 num -= 3; // num = 12 num *= 2; // num = 24 num /= 6; // num = 4 num %= 3; // num = 1

program:-
```
#include <stdio.h>
#include <stdbool.h>

int main() {
    // Arithmetic Operators
    int num1 = 10;
    int num2 = 3;
    int sum = num1 + num2;
    int difference = num1 - num2;
    int product = num1 * num2;
    int quotient = num1 / num2;
    int remainder = num1 % num2;

    printf("Arithmetic Operators:\n");
    printf("Sum: %d\n", sum);
    printf("Difference: %d\n", difference);
    printf("Product: %d\n", product);
    printf("Quotient: %d\n", quotient);
    printf("Remainder: %d\n", remainder);

    // Relational Operators
    int age = 25;
    bool isStudent = false;
    bool isTeenager = (age >= 13 && age <= 19);
    bool isAdult = (age >= 18 || isStudent);

    printf("\nRelational Operators:\n");
    printf("Is teenager: %d\n", isTeenager);
    printf("Is adult: %d\n", isAdult);

    // Logical Operators
    bool isNotStudent = !isStudent;
    bool result = (isTeenager && isNotStudent);

    printf("\nLogical Operators:\n");
    printf("Is not a student: %d\n", isNotStudent);
    printf("Result: %d\n", result);

    // Bitwise Operators
    int a = 5; // binary: 0101
    int b = 3; // binary: 0011

    int bitwiseAnd = a & b;
    int bitwiseOr = a | b;
    int bitwiseXor = a ^ b;
    int bitwiseNot = ~a;
    int leftShift = a << 2;
    int rightShift = a >> 1;

    printf("\nBitwise Operators:\n");
    printf("Bitwise AND: %d\n", bitwiseAnd);
    printf("Bitwise OR: %d\n", bitwiseOr);
    printf("Bitwise XOR: %d\n", bitwiseXor);
```

```
    printf("Bitwise NOT: %d\n", bitwiseNot);
    printf("Left Shift: %d\n", leftShift);
    printf("Right Shift: %d\n", rightShift);

    // Assignment Operators
    int num = 10;
    num += 5;
    num -= 3;
    num *= 2;
    num /= 6;
    num %= 3;

    printf("\nAssignment Operators:\n");
    printf("Number: %d\n", num);

    return 0;
}
```

3.1 Decision Making:

- If Statement: The if statement is a conditional statement that allows the execution of a block of code only if a specified condition is true. If the condition is false, the code block is skipped.
- If else Statement: The if else statement is an extension of the if statement. It allows the execution of one block of code if a condition is true, and a different block of code if the condition is false.
- Nesting of if-else: Nesting of if-else statements refers to the use of one if-else statement inside another if-else statement. This allows for the execution of different blocks of code based on multiple conditions.

3.2 Branching:

- Switch Statement: The switch statement is used to select one of many code blocks to be executed. It evaluates a variable or expression and matches it with the value of different cases. The code block associated with the matching case is executed.

3.3 Looping:

- While Loop: The while loop repeatedly executes a block of code as long as a specified condition is true. It checks the condition before executing the code block.

- Do-While Loop: The do-while loop is similar to the while loop, but it checks the condition after executing the code block. This guarantees that the code block is executed at least once, even if the condition is initially false.
- For Loop: The for loop is used when the number of iterations is known or can be determined. It consists of an initialization, a condition, and an increment/decrement section. The code block is executed as long as the condition is true.

3.4 Ternary Operator: The ternary operator is a shorthand way of writing if-else statements in a single line. It takes three operands: a condition, a value if the condition is true, and a value if the condition is false.
3.5 Go to Statement: The goto statement transfers the control of the program to a labeled statement within the same function. It allows for non-linear control flow, but it is generally discouraged due to its potential to make code harder to understand and maintain.
3.6 Use of break and continue statements:

- The break statement is used to terminate the execution of a loop or switch statement. It is typically used to exit a loop early if a certain condition is met.
- The continue statement is used to skip the rest of the current iteration of a loop and move to the next iteration. It is typically used to skip specific iterations based on certain conditions.

3.1 Decision Making:

- If Statement: The if statement is used to execute a block of code if a certain condition is true.

Example program:

```
#include <stdio.h>

int main() {
    int num = 10;

    if (num > 0) {
        printf("The number is positive.\n");
    }

    return 0;
}
```

- If else Statement: The if else statement is used to execute different blocks of code based on different conditions.

Example program:
```
#include <stdio.h>

int main() {
    int num = 10;

    if (num > 0) {
        printf("The number is positive.\n");
    } else {
        printf("The number is not positive.\n");
    }

    return 0;
}
```

- Nesting of if-else: The nesting of if-else statements allows the execution of different blocks of code based on multiple conditions.

Example program:
```
#include <stdio.h>

int main() {
    int num = 10;

    if (num > 0) {
        printf("The number is positive.\n");
    } else {
        if (num < 0) {
            printf("The number is negative.\n");
        } else {
            printf("The number is zero.\n");
        }
    }

    return 0;
}
```

3.2 Branching:

- Switch Statement: The switch statement allows the execution of different blocks of code based on the value of a variable.

Example program:

```
#include <stdio.h>

int main() {
    int choice;

    printf("Enter a number between 1 and 3: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("You chose option 1.\n");
            break;
        case 2:
            printf("You chose option 2.\n");
            break;
        case 3:
            printf("You chose option 3.\n");
            break;
        default:
            printf("Invalid choice.\n");
    }

    return 0;
}
```

3.3 Looping:

- While Loop: The while loop repeatedly executes a block of code as long as a certain condition is true.

Example program:

```
#include <stdio.h>

int main() {
    int count = 1;

    while (count <= 5) {
        printf("Count: %d\n", count);
        count++;
    }

    return 0;
}
```

- Do-While Loop: The do-while loop first executes a block of code and then checks the condition. It continues to execute the block as long as the condition is true.

Example program:

```c
#include <stdio.h>

int main() {
    int count = 1;

    do {
        printf("Count: %d\n", count);
        count++;
    } while (count <= 5);

    return 0;
}
```

- For Loop: The for loop allows the repeated execution of a block of code based on a specified condition.

Example program:

```c
#include <stdio.h>

int main() {
    for (int count = 1; count <= 5; count++) {
        printf("Count: %d\n", count);
    }

    return 0;
}
```

3.4 Ternary Operator: The ternary operator is a shorthand way of writing if-else statements in a single line.

Example program:

```c
#include <stdio.h>

int main() {
    int num = 10;

    int result = (num > 0) ? num : -num;
    printf("Result: %d\n", result);

    return 0;
}
```

3.5 Go to Statement: The goto statement allows jumping to a specific labeled statement within the program.
Example program:

```
#include <stdio.h>

int main() {
    int num = 10;

    if (num > 0) {
        goto positive;
    } else {
        goto negative;
    }

    positive:
    printf("The number is positive.\n");
    goto end;

    negative:
    printf("The number is negative.\n");
    goto end;

    end:
    return 0;
}
```

3.6 Use of break and continue statements:

- The break statement is used to terminate the execution of a loop or switch statement.
- The continue statement is used to skip the rest of the current iteration of a loop and move to the next iteration.

Example program:

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

```c
int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            break;      // Terminate the loop when i is 3
        }

        printf("i: %d\n", i);
    }

    printf("\n");

    for (int j = 1; j <= 5; j++) {
        if (j == 3) {
            continue;   // Skip the rest of the iteration when j is 3
        }

        printf("j: %d\n", j);
    }

    return 0;
}
```