Dr. SNS RAJALAKSHMI COLLEGE OF ARTS AND SCIENCE

(AUTONOMOUS)

Accredited by NAAC (Cycle- III) with 'A+' Grade

DEPARTMENT OF B.SC CS (GCD)

23UCU401 – PROGRAMMING IN C
UNIT- I

Introduction to C Programming

C Programming Language Overview

- C is a powerful and versatile programming language developed in the early 1970s by Dennis Ritchie at Bell Labs.
- It's known for its efficiency, flexibility, and portability, making it one of the most widely used programming languages.

Popularity and Applications

- C is used in various domains, including system programming, embedded systems, game development, and more.
- Many operating systems, like UNIX, are written in C, demonstrating its system-level capabilities.

Structured Programming

- C follows a structured programming approach, which emphasizes the use of functions, control structures, and modular design.
- This approach simplifies program development and maintenance.

C vs. Other Programming Languages

- C is often compared to other languages like C++, Java, and Python, highlighting its strengths and differences.

Basic Syntax

- C programs consist of functions, variables, data types, and statements.
- Understanding the syntax is crucial for writing correct and efficient code.

Compilers and IDEs

- To develop C programs, you need a C compiler (e.g., GCC) and an Integrated Development Environment (IDE) like Code::Blocks or Visual Studio.

Portability

- C code can be compiled and run on various platforms with minimal modifications, thanks to its portability.

The Importance of Learning C

- Learning C serves as a solid foundation for understanding other programming languages and low-level system interactions.

2. Writing Your First C Program

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

- Explanation:
    - The program structure consists of the main() function.
    - The program prints "Hello, World!" using printf() and returns 0.

3. Compilation and Execution

- Compilation involves converting your C source code into an executable.
- Execution is the process of running the compiled program.
- Popular C development environments include Code::Blocks, Visual Studio, and GCC.

Compilation is the process of converting the source code of a C program into machine code. Machine code is the only language that the computer can understand and execute.

The compilation process in C involves four steps:

1. Preprocessing: The preprocessor removes comments from the source code, expands macros, and includes header files.
2. Compiling: The compiler translates the preprocessed source code into assembly language. Assembly language is a low-level language that is specific to the computer architecture.
3. Assembling: The assembler converts the assembly language code into object code. Object code is machine code that is specific to the computer operating system.
4. Linking: The linker combines the object code file with the library files to create an executable file. The executable file can be run on the computer to produce the desired output.

Execution is the process of running the executable file to produce the desired output.

4. Receiving Input

To receive input in C, you can use the `scanf()` function. The `scanf()` function reads formatted input from the standard input stream, which is typically the keyboard.

The `scanf()` function takes two arguments:

1. A format string, which specifies the type of data to be read.
2. A pointer to a variable where the read data will be stored.

For example, to read an integer from the user, you would use the following code:

```
int number;
scanf("%d", &number);
```

The `%d` format specifier tells the `scanf()` function to read an integer. The `&` operator tells the `scanf()` function to store the read data in the `number` variable.

You can also use the `scanf()` function to read other types of data, such as floating-point numbers, characters, and strings

For example, to read a character from the user, you would use the following code:

```
char character;
scanf("%c", &character);
```

To read a string from the user, you would use the following code:

```
char string[100];
scanf("%s", string);
```

The `%s` format specifier tells the `scanf()` function to read a string. The `string` variable is an array of characters that can store up to 100 characters.

5. C Instructions

C instructions are the commands in a C program that tell the compiler what to do. They are the basic building blocks of C programs, and they are used to perform all of the tasks that a C program can do.

There are many different types of C instructions, but they can be broadly categorized into the following groups:

- Declaration instructions: These instructions are used to declare variables, functions, and other types of data.
- Execution instructions: These instructions are used to perform operations on data, such as arithmetic operations, logical operations, and comparisons.
- Control flow instructions: These instructions are used to control the flow of execution of a program, such as by using conditional statements and loops.
- Input/output instructions: These instructions are used to read and write data to and from external devices, such as the keyboard and the screen.

6. Control Instructions in C

Control instructions in C are used to control the flow of execution of a program. They allow programmers to make decisions based on the value of variables or expressions, and to loop through blocks of code until a certain condition is met.

The most common control instructions in C are:

- if: Executes a block of code if a condition is met.
- else: Executes a block of code if an if condition is not met.
- switch: Executes a block of code based on the value of a variable.
- for: Executes a block of code a certain number of times.
- while: Executes a block of code while a condition is met.
- do-while: Executes a block of code at least once, then executes it again while a condition is met.

Control instructions are essential for writing complex and powerful C programs. They allow programmers to create programs that can make decisions, repeat tasks, and process data in a variety of ways.

7. The Decision Control Structure - The if Statement

The `if` statement is the simplest decision control structure in C. It allows programmers to execute a block of code only if a certain condition is met.

The syntax for the `if` statement is as follows:

```
if (condition) {
// Code to be executed if the condition is met
}
```

The `condition` can be any Boolean expression, such as a comparison between two variables, a check to see if a variable is greater than zero, or a call to a function that returns a Boolean value.

If the condition evaluates to `true`, the code inside the `if` block will be executed. Otherwise, the code inside the `if` block will be skipped.

Here is an example of an `if` statement:

```
int number = 10; if (number > 5) {
printf("The number is greater than 5.\n");
}
```

In this example, the `if` statement will check to see if the variable `number` is greater than 5. If it is, the code inside the `if` block will be executed, which will print the message "The number is greater than 5." to the console. Otherwise, the code inside the `if` block will be skipped.

`if` statements can be nested to create more complex decision-making logic. For example, the following code shows a nested `if` statement that checks to see if a number is even or odd:

```
int number = 10;
if (number % 2 == 0) {
printf("The number is even.\n");
} else {
printf("The number is odd.\n");
}
```

This code will first check to see if the number is divisible by 2. If it is, the code will print the message "The number is even." to the console. Otherwise, the code will print the message "The number is odd." to the console.

8. The if-else Statement

The `if-else` statement is a decision control structure in C that allows programmers to execute different blocks of code based on the value of a Boolean expression.

The syntax for the `if-else`

```
if (condition) {
// Code to be executed if the condition is met
} else {
// Code to be executed if the condition is not met
}
```

The `condition` can be any Boolean expression, such as a comparison between two variables, a check to see if a variable is greater than zero, or a call to a function that returns a Boolean value.

If the condition evaluates to `true`, the code inside the `if` block will be executed. If the condition evaluates to `false`, the code inside the `else` block will be executed.

`if-else` statements can be used to write a variety of different programs, such as programs that:

- Check for errors and take corrective action
- Determine the best course of action based on the current state of the program
- Allow the user to select from different options

`if-else` statements are a powerful tool that can be used to write complex and powerful C programs.

9. Use of Logical Operators

Logical operators in C are used to combine multiple Boolean expressions into a single Boolean expression. They can be used to create complex decision-making logic in C programs.

There are three logical operators in C:

- `&&`: Logical AND
- `||`: Logical OR
- `!`: Logical NOT

The logical AND operator (`&&`) returns `true` only if both of its operands evaluate to `true`. Otherwise, it returns `false`.

The logical OR operator (`||`) returns `true` if either of its operands evaluates to `true`. Otherwise, it returns `false`.

The logical NOT operator (`!`) inverts the truth value of its operand. For example, if the operand evaluates to `true`, the logical NOT operator will return `false`, and vice versa.

Logical operators can be used to create a variety of different expressions, such as:

- `(x > 5) && (y < 10)`: This expression will evaluate to `true` only if both `x` is greater than 5 and `y` is less than 10.
- `(x == 10) || (y == 20)`: This expression will evaluate to `true` if either `x` is equal to 10 or `y` is equal to 20.
- `!(x == 0)`: This expression will evaluate to `true` only if `x` is not equal to 0.

Here is an example of how to use logical operators in a C program:

```
int main() {
  int x = 5;
  int y = 10;
  int z = 20;

  // Check if x is greater than 5 and y is less than 10
  if (x > 5 && y < 10) {
    printf("x is greater than 5 and y is less than 10.\n");
  }

  // Check if x is equal to 10 or y is equal to 20
  if (x == 10 || y == 20) {
    printf("x is equal to 10 or y is equal to 20.\n");
  }

  // Check if x is not equal to zero
  if (!x == 0) {
    printf("x is not equal to zero.\n");
  }

  return 0;
}
```

10. The Conditional Operators (Ternary Operator)

The conditional operator in C, also known as the ternary operator, is a powerful tool that can be used to write concise and efficient code. It allows you to evaluate a condition and return one of two values based on the result.

condition ? value_if_true : value_if_false;

The `condition` can be any Boolean expression, such as a comparison between two variables, a check to see if a variable is greater than zero, or a call to a function that returns a Boolean value.

If the `condition` evaluates to `true`, the `value_if_true` expression will be evaluated and returned. Otherwise, the `value_if_false` expression will be evaluated and returned.

Here is an example of how to use the conditional operator:

```
int x = 5;
int y = 10;
// Check if x is greater than 5 and return 10 if it is, otherwise return 5.
int result = x > 5 ? 10 : 5;
 printf("result = %d\n", result);
```