|  |  |
| --- | --- |
| CL1002  P***rogramming Fundamentals Lab*** | Lab 03  Basic Decision Structures |

# National University of Computer and Emerging Sciences

Fall 2024.

# Aims and Objectives

**Aim:**

The aim of learning basic decision structures in C is to understand and implement logical decision-making in programs. This involves using conditions to control the flow of execution, allowing programs to make choices based on different criteria, enhancing their interactivity, functionality, and responsiveness.

**Objectives:**

1. **Understand the Concept of Decision Structures:**
   * Grasp the importance of decision-making in programming and how it helps in controlling the flow of a program.
2. **Learn Basic Decision-Making Statements:**
   * Familiarize yourself with the primary decision structures in C, such as if, if-else, and switch statements.
3. **Implement if and if-else Statements:**
   * Learn how to use if statements to execute code blocks based on a condition.
   * Use if-else statements to handle multiple branches, allowing different actions based on whether a condition is true or false.
4. **Use switch Statements for Multiple Choices:**
   * Learn how to use switch statements for scenarios involving multiple possible cases, improving code readability and organization over multiple if-else conditions.
5. **Understand Relational and Logical Operators:**
   * Gain proficiency in using relational (==, !=, <, >, <=, >=) and logical operators (&&, ||, !) to form complex conditions within decision structures.

# Introduction

Decision structures are fundamental building blocks in programming that allow programs to make choices based on certain conditions. In C, decision structures enable you to control the flow of execution, making your programs dynamic and responsive to different inputs and scenarios.

At the core of decision-making in C are statements like if, if-else, and switch, which provide mechanisms to execute specific blocks of code depending on whether certain conditions are true or false. These structures allow the program to "decide" what actions to take, making them essential for building interactive applications, performing calculations based on user input, managing error handling, and much more.

Understanding and effectively using decision structures is a critical skill for any programmer. They form the basis for writing logical, efficient, and flexible code that can adapt to varying requirements. Whether you are developing simple applications or complex systems, mastering decision structures will enhance your ability to create programs that can intelligently respond to different situations.

# Conditional Statements

In C programming there are decision making statements, we need these kinds of statements because while programming we often need to make a lot of decisions. Lets take an example of a traffic signal management as show below

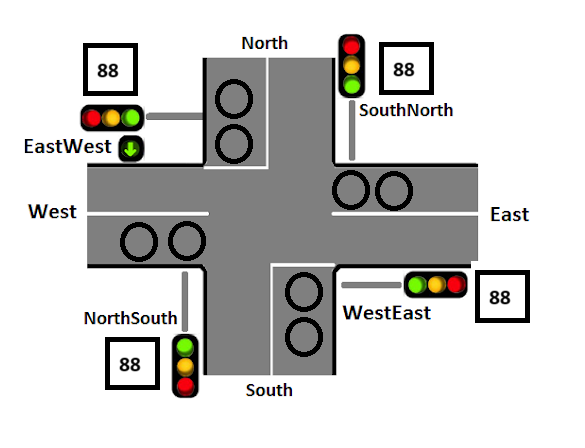


Figure 1. Example of a 4-Way smart traffic signal

In most countries there are vehicle sensors embedded in the road that detect how many cars are present in that lane. In the above-mentioned example these sensors are represented via black circles. If both circles are occupied by cars in that lane the time would be reduced to allow a smoother flow of traffic else if only one of the sensors is covered in that lane, then the timer would engage normally and start counting downwards until the counter value hits zero. Otherwise, if no sensors detects a vehicle then the signal would remain red.

# Example 1

In some cases, we need to execute this block of code

1. If the number is odd
2. {
3. //Execute some code if both sensors are detecting a vehicle
4. }
5. else
6. {
7. //Execute some code
8. }

Otherwise, we want to execute this block

1. If the number is odd
2. {
3. //Execute some code
4. }
5. else
6. {
7. //Keep the signal light red if sensors detect no vehicle
8. }

In C, an if/else statement specifies that one block of code should be executed if a condition is true, and another block should be executed if that condition is false.

To write meaningful if/else statements, it is important to know operators which allow us to compare two expressions and produce a Boolean outcome.

In C, however, there are no distinct values for true or false, instead a false condition is denoted by a number 0 and anything which is non-zero is considered true.

|  |  |
| --- | --- |
| expr1 == expr 2 | This condition checks if expr1 is equal to expr2 |
| expr1 != expr 2 | This condition checks if expr1 is not equal to expr2 |
| expr1 < expr 2 | This condition checks if expr1 is less than expr2 |
| expr1 <= expr 2 | This condition checks if expr1 is less than or equal to expr2 |
| expr1 > expr 2 | This condition checks if expr1 is greater than expr2 |
| expr1 >= expr 2 | This condition checks if expr1 is greater than or equal to expr2 |
| !expr1 | This condition checks the logical NOT of expr1 |
| expr1 && expr 2 | This condition checks the logical AND of expr1 and expr2 |
| expr1 || expr 2 | This condition checks the logical OR of expr1 and expr2 |

Figure 2. Use cases for if-else if- else statement

In 'C' programming conditional statements are possible with the help of the following two constructs:

1. If statement

2. If-else statement

It is also called branching as a program decides which statement to execute based on the result of the evaluated condition.

# IF-STATEMENT

An if statement consists of a conditional expression followed by one or more statements.

If the conditional expression evaluates to true, then the block of code inside the if statement will be executed. If the conditional expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

**Syntax:** The syntax of an if statement in C programming language is:

**If (condition)**

**{**

**//statements;**

**}**

**Flowchart**

Diagram

Description automatically generated

# Example 2

**Checking if the number input by user is 0 or not. If it's 0 then print Zero else print non-zero**

1. #include <stdio.h>
2. **int** main() {
3. **int** num1, num2;
4. printf (“Enter two integers \n”);
5. scanf (“%d%d”,&num1,&num2);
6. if(num2!=0){
7. printf(“num1/num2 = %d\n”,num1/num2);
8. }
9. return 0;
10. }

# IF-ELSE STATEMENT

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

**Syntax:** The syntax of an if...else statement in C programming language is:

**If (condition)**

**{**

**//statements;**

**}**

**else**

**{**

**//statements;**

**}**

S

S

ss

# Example 3

Check if one number is greater than the other.

1. #include <stdio.h>
2. **int** main() {
3. **int** num1, num2;
4. printf (“Enter two integers \n”);
5. scanf (“%d%d”,&num1,&num2);
6. if(num2>0){
7. printf(“num2 = %d is greater\n”,num2);
8. }else{
9. printf(“num1 = %d is greater ”,num1);
10. }
11. return 0;
12. }

# IF-ELSE IF-ELSE STATEMENT

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single **if...else if** statement.

When using **if, else if, else** statements there are few points to keep in mind:

* An **if** can have zero or one else's and it must come after any **else ifs**.
* An **if** can have zero to many **else if's** and they must come before the **else**.
* Once an **else if** succeeds, none of the remaining **else if's** or **else's** will be tested.

# Example 4

**Syntax:** The syntax of an if...else statement in C programming language is:

1. #include <stdio.h>
2. **int** main() {
3. **int** num1;
4. printf (“Enter value =\n”);
5. scanf (“%d”,&num1);
6. if(num1>0){
7. printf(“num1 = %d is positive\n”,num1);
8. }else if (num1<0){
9. printf(“num1 = %d is negative ”,num1);
10. }else {
11. printf(“num1 = %d is zero ”,num1);
12. }
14. return 0;
15. }

# Switch-Case-Statements

Another way that programs can make decisions is to use switch/case. The syntax of switch/case is shown in the figure below.

**switch (selection expression) {**

**case 1:**

**//statement**

**break;**

**case 2:**

**//statement**

**break;**

**default:**

**//statement**

**}**

Here, when the execution arrow reaches the switch statement, the selection expression—in parenthesis after the keyword switch—is evaluated to a value.

This value is then used to determine which case to enter. The execution arrow then jumps to the corresponding case—the one whose label (the constant immediately after the keyword case) matches the selection expression’s value. If no label matches, then the execution arrow jumps to the default case if there is one, and to the closing curly brace of the switch if not.

✔

**switch (selection expression)**

**int / char**

**float / string**

Figure 3: Switch case data type requirement

The following rules apply to a **switch** statement:

* The **expression** used in a **switch** statement must have an integral or enumerated type or be of a class type in which the class has a single conversion function to an integral or enumerated type.
* You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.

**Switch statement is better than if else statement**

A switch statement is **usually more efficient than a set of nested ifs**

Switch statement acts as a substitute for a long **if-**else-if ladder that is used to test a list of cases.

Chart, bubble chart

Description automatically generated

# Example 5

1. #include <stdio.h>
2. **int** main() {
3. **int** day; // Variable to store the user's input
4. **printf**("Enter a number (1-7) to get the corresponding day of the week: ");
5. **scanf**("%d", &day);
6. // Switch case to determine the day of the week
7. **switch** (day) {
8. **case** 1:
9. printf("Monday\n");
10. **break**;
11. **case** 2:
12. printf("Tuesday\n");
13. **break**;
14. **case** 3:
15. printf("Wednesday\n");
16. **break**;
17. **case** 4:
18. printf("Thursday\n");
19. **break**;
20. **case** 5:
21. printf("Friday\n");
22. **break**;
23. **case** 6:
24. printf("Saturday\n");
25. **break**;
26. **case** 7:
27. printf("Sunday\n");
28. **break**;
29. **default**:
30. printf("Invalid input! Please enter a number between 1 and 7.\n");
31. }
32. return 0;
33. }

Note on every case (except default) we have to use the break keyword as the program will keep checking for other cases even if the first case is verified.