
# week 2_1:

# conditional statements

rajin teaches programming

**- OVERVIEW:**

Welcome to **series 2.** In this part, we'll dive into **control structures,** specifically focusing on **if-else statements** and **switch-case** statements. Understanding control structures is crucial for controlling the **flow** of your **program** based on **conditions** and making decisions **dynamically.**

## INTRODUCTION TO CONDITIONAL STATEMENTS:

**Conditional statements** are **fundamental** to programming as they **allow** us to execute certain **blocks** of code based on whether a **condition** is **true** or **false.** They enable our programs to make **decisions** dynamically, **adapting** to different scenarios.
For example, we want to build a program that checks if a student has passed or failed an exam based on their score. We can use a conditional statement to achieve this:

Developer - test.c

```c
#include<stdio.h>

int main() {

    int score = 75;

    if (score >= 50) {

            printf("Congratulations! You have passed the exam.");
        }
        else {

            printf("Sorry, you have failed the exam. Better luck next time.");
         }

    return 0;
}
```

*an example of a conditional statement*

## IF-ELSE STATEMENTS:

**- UNDERSTANDING IF-ELSE:**

**If-else** statements allow us to execute **certain** blocks of code based on whether a **condition** is **true** or **false.**

- **SYNTAX:**

IF-ELSE SYNTAX

```
if (condition) {
        // Code to execute if condition is true
}
else {
        // Code to execute if condition is false
}
```

Developer - vote.c

```c
#include<stdio.h>

int main() {

    int age = 20;

    if (age >= 18) {

            printf("You are eligible to vote.");
        }
        else {

            printf("You are not eligible to vote.");
        }

    return 0;
}
```

*another example of an if-else statement*

Inside the **condition** part of an if block, we make use of **comparison operators.** These **operators** help us evaluate an **expression** to either **TRUE (1)** or **FALSE (0).**

## COMPARISON OPERATORS:

**Comparison operators** are used to compare **two values** and return a **boolean result** [**TRUE (1)** or **FALSE (0)**].

Here some comparison operators (**other** combinations may exist):

| OPERATOR | DESCRIPTION | EXAMPLE |
|---|---|---|
| == | EQUAL TO | if (x==10) |
| != | NOT EQUAL TO | if (y!=0) |
| < | LESS THAN | if (z<5) |
| > | GREATER THAN | if (a>b) |
| <= | LESS THAN OR EQUAL TO | if (x<=y) |
| >= | GREATER THAN OR EQUAL TO | if (a>=b) |

## LOGICAL OPERATORS:

**Logical operators** are used to **combine multiple conditions** and determine the **overall boolean** [**TRUE (1)** or **FALSE (0)**] value**.**

### – LOGICAL AND ("&&"):

The logical AND operator returns true if **both** operands are **true.**

Developer - vote.c

```c
#include<stdio.h>

int main() {

    int age = 20;
    int income = 50000;

    if (age >= 18 && income >= 30000) {

            printf("You are eligible for a loan.");
        }
        else {

            printf("You are not eligible for a loan.");
         }

    return 0;
}
```

*an example of logical AND being used*

### – LOGICAL OR ("||"):

The logical OR operator returns true if **at least one** of the operands is **true.**

Developer - overtime.c

```c
#include<stdio.h>

int main() {

    int hours_worked = 45;

    if (hours_worked > 40 || hours_worked < 60) {

        printf("You are eligible for overtime pay.");
    }

    return 0;
}
```

*an example of logical OR being used*

**– LOGICAL NOT ("!"):**

The logical NOT operator returns the **opposite** of the **operand's value.**

Developer - flag.c

```c
#include<stdio.h>

int main() {

    int flag = 0;

    if (!flag) {

        printf("Flag is not set.");
    }

    return 0;
}
```

*an example of logical NOT being used*

## NESTED IF STATEMENTS:

**Nested if-statements** are if-statements **inside** other if statements. They allow for more **complex conditional logic** by evaluating **multiple conditions.** For example:

Developer - nested.c

```c
#include<stdio.h>

int main() {

    int x = 5;
    int y = 10;

    if (x > 0) {

        if (y > 0) {

            printf("Both x and y are positive.");
        }
    }

    return 0;
}
```

*an example of a NESTED IF statement*

## SWITCH-CASE STATEMENTS:

### - UNDERSTANDING SWITCH-CASE:

**Switch-case statements** allow us to select **one** of **many code blocks** to execute based on the **value** of a **variable** or **expression.** Think of it as an **alternative** to **long chains** of **if-else** statements.

### - SYNTAX:

SWITCH-CASE SYNTAX

```
switch (expression) {
      case value1:

            // Code to execute if expression is equal to value1
            break;

      case value2:

            // Code to execute if expression is equal to value2
            break;

      default:

            // Code to execute if expression doesn't match any case
    }
```

Developer - switchcase.c

```c
#include<stdio.h>

int main() {

    char grade = 'B';

    switch (grade) {

        case 'A':

            printf("Excellent!");
            break;

        case 'B':

            printf("Good work!");
            break;

        case 'C':

            printf("Keep improving.");
            break;

        default:

            printf("Invalid grade.");
    }

    return 0;
}
```

*another example of a switch-case statement*

## CONCLUSION:

Understanding control structures/conditional statements like if-else and switch-case statements is **essential** for writing **dynamic** and **flexible** programs. By mastering these concepts, you'll gain the ability to **control** the **flow** of your program based on **conditions**, making your programs **more powerful** and **responsive.**

Keep practicing and experimenting with different scenarios to reinforce your understanding. If you have any questions or need further clarification, as always, don't hesitate to ask.


# next class 2_2:

# loops

rajin teaches programming

*reach out for classes at: rajin.khan2001@gmail.com*