Patrician College of Arts and Science

Department of Computer Applications

Programming in C++

SAZ3A

Odd Semester

Presented By
Josephine Shanthi

https://www.patriciancollege.ac.in/

DECISION MAKING AND BRANCHING:

- We have a number of situations where, we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly. C++ has the decision-making and branching statements. They are
  - if statement
  - switch statement
  - Conditional operator statement
  - goto statement

THE IF STATEMENT:

- The if statement is a powerful decision-making statement and is used to control the flow of execution of statements. It takes the following form:

  if (test expression)

- It allows the computer to evaluate the expression first and then depending on whether the value of the expression is ‘true’ (or non-zero) or ‘false’ (or zero), it transfers the control to a particular statement.

- The if statement may be implemented in different forms depending on the complexity if conditions to be tested. The different forms are:
  - Simple if statement.
  - If……else statement.
  - Nested if….else statement
  - else …if ladder

THE SWITCH STATEMENT

- This is a multiple-branching statement where, based on condition, the control is transferred to one of the many possible points. The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.

THE CONDITIONAL OPERATOR

- The general form of use of conditional operator is

  Conditional expr ? expr1 : exp2;

- The conditional expr is evaluated first. If the result is nonzero (true), then the expression expr1 is evaluated and returned as the value of the conditional expression. If it is zero (false), then the expression expr2 is evaluated and becomes the value of the expression. Only one of the expressions either expr1 or expr2 is evaluated.

goto STATEMENT

- C++ supports the **goto** statement to branch unconditionally from one point to another in the program. A goto statement breaks the normal sequential execution of the program.
- Using goto statement sometimes lead to infinite loop. This condition should be avoided.
- The goto statement is used to transfer the control out of a loop, when certain peculiar conditions are encountered.
- We should try to avoid using goto statements as far as possible.

DECISION MAKING AND LOOPING:

- C++ has the decision-making and looping statements. They are
  - while statement
  - do… while statement
  - for statement

THE WHILE STATEMENT:

- This is a loop structured, and *entry-controlled* one.
- Syntax

```
while (condition)
 {
        // code block to be executed
}
```

THE DO-WHILE STATEMENT:

- The do-while is an *exit-controlled* loop. Based on a condition, the control is transferred back to a particular point in the program.
- Syntax

```
do
{
        // code block to be executed
}while (condition);
```

THE for STATEMENT

- The **for** is an *entry-controlled* loop and is used when an action is to be repeated for a predetermine number of times.
- Syntax

```
for (statement 1; statement 2; statement 3)
{
        // code block to be executed
}
```

- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed

Jumping out of a Loop:

- An early exit from a loop can be accomplished by using the **break** statement
- When a **break** statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. The **break** will exit only a single loop.

Skipping a part of a loop:

- C++ supports the **continue** statement. The continue statement causes the loop to be continued with the next iteration after skipping any statements in between.

Jumping out of a program:

- We can jump out of a program by using the library function **exit()**. The use of **exit()** function requires the inclusion of the header file <stdlib.h>. In case due to some reason, we wish to break out of a program and return to the operating system, we can use this statement.

FUNCTIONS:

- A function is a self-contained block of code that performs a particular task
- ADVANTAGES:
- It facilitates top-down modular programming
- The length of a source program can be reduced by using functions at appropriate places.
- It is easy to locate and isolate a faulty function for further investigations.
- A function may be used by many other programs
- You can pass data, known as parameters, into a function.
- Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

ELEMENTS OF USER-DEFINED FUNCTIONS

- In order to make use of a user-defined function, we need to establish
  - Function definition
  - Function call
  - Function declaration

Function definition

- A function definition, also known as function implementation shall include the following elements:

1. function name	|	
2. function type	|	Function header
3. list of parameters	|	
4. local variable declaration	}	
5. function statements and	}	Function body
6. a return statement.	}	

- The ***function header*** consists of three parts: the function type (also known as return type), the function name and the formal parameter list.

- The ***function type*** specifies the type of the value that the function is expected to return the program calling the function. If the return type is not explicitly specified, C will assume that it is an integer type.

- The ***parameter list*** declares the variables that will receive the data sent by calling program. They serve as input data to the function to carry out the specified task. Since they represent actual input values, they are often referred to as ***formal parameters***. These parameters can also be used to send values to the calling programs. The parameters are also known as arguments.

- The ***function body*** contains the declarations and statements necessary for performing the required task. If a function does not return any value we can omit the return statement.

- When a function reaches its ***return*** statement, the control is transferred back to the calling program. In the absence of a return statement, the closing brace acts as a void return.

FUNCTION CALLS:

- A function can be called by simply using the function name followed by a list of ***actual parameters*** *(or arguments)*, if any, enclosed in parentheses.
- When a function is called, the compiler uses the template to ensure that proper arguments are passed, and the return value is treated correctly. Any violation in matching the arguments or the return types will be caught by the compiler at the time of compilation itself.

- Functions can be invoked in two ways:
  - Call by Value
    - This is the default way of calling a function. The parameters are passed to a function are called actual parameters and the parameters received by the function are called formal parameters.
  - Call by Reference
    - Both the actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.
    - When we call a function by passing the addresses of actual parameters then this way of calling the function is known as call by reference. In call by reference, the operation performed on formal parameters, affects the value of actual parameters because all the operations performed on the value stored in the address of actual parameters.

- A function, depending on whether arguments are present or not and whether a value is returned or not, may belong to one of the following categories:
  1. Functions with no arguments and no return values.
  2. Functions with arguments and no return values.
  3. Functions with arguments and one return value.
  4. Functions with no arguments but returns a value.
  5. Functions that return multiple values.

RECURSION:

- Recursion is a special process, where a function calls itself.

Thank you

https://www.patriciancollege.ac.in/