Control Structures

A computer can process a program in one of the following ways:

- **In sequence:** in which the program executes the statements from first to last. The first statement is executed, then the second, then the third, and so on, until the program reaches its end and terminates;
- **Selectively (Conditional)**, by making a choice, which is also called a branch;
- **Repetitively (Loop)**, by executing a statement over and over, using a structure called a loop; or by calling a function.

Figure 1 illustrates the first three types of program flow. The two most common control structures are selection and repetition. In selection, the program executes particular statements depending on some condition(s). In repetition, the program repeats particular statements a certain number of times based on some condition(s).

Thus, **Control structures** are portions of program code that contain statements within them and, depending on the circumstances, execute these statements in a certain way.

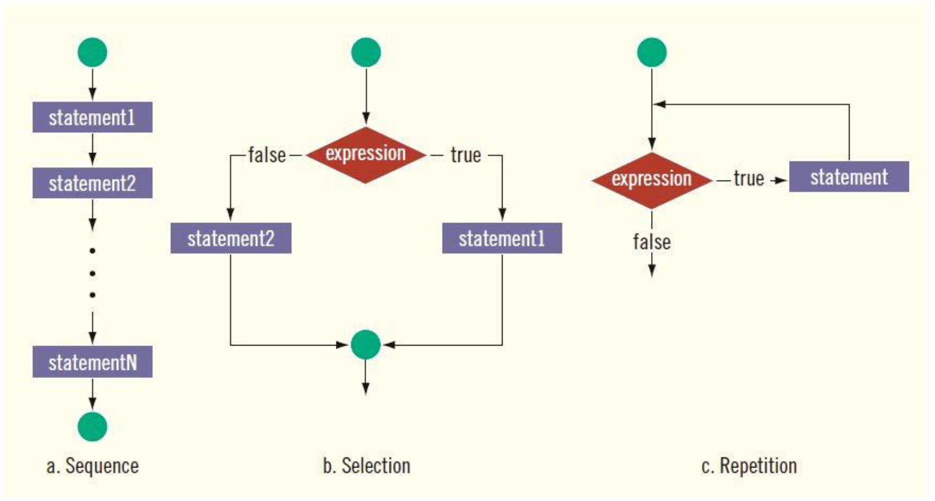
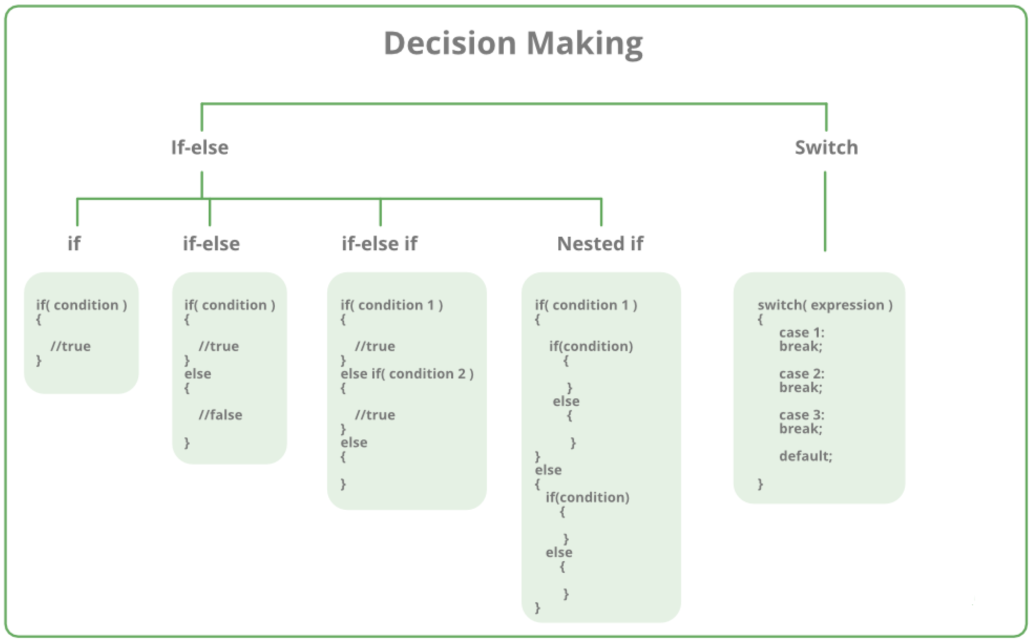


Figure 1: Flow of execution

1- Conditionals: Decision Making

There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.

In order for a program to change its behavior depending on the input, there must a way to test that input. Conditionals allow the program to check the values of variables and to execute (or not execute) certain statements. C++ has if and switch-case conditional structures. Figure 2 shows the decision making statements available in C/C++ are:



```
if( condition )
{
   //true
}
```

```
if( condition )
{
   //true
}
else
{
   //false
}
```

```
if( condition 1 )
{
   //true
}
else if( condition 2 )
{
   //true
}
else
{
}
```

```
if( condition 1 )
{
   if(condition)
     {
     }
    else
     {
     }
}
else
{
  if(condition)
    {
    }
   else
    {
    }
}
```

```
switch( expression )
{
    case 1:
    break;

    case 2:
    break;

    case 3:
    break;

    default;
}
```

Figure 2: Decision making statements available in C/C++

1- Selection: *if* and *if...else , if, if-else and else if statements*

Although there are only two logical values, **true** and **false**, they turn out to be extremely useful because they permit programs to incorporate decision making that alters the processing flow. In C++, there are two selections, or branch control structures: **if statements** and the **switch structure**. This section discusses how if and if...else statements can be used to create

- One-way selection: **If**
- Two-way selection. **If …. else**
- Multiple selections. **Else if**

- One-way selection: (If)

In C++, one-way selections are incorporated using the if statement.
The **syntax** of one-way selection is:

```
if( expression )
{
      // Statements to execute if condition is true
      statement1;
      statement2;
      …
}
```

Note the elements of this syntax. It begins with the reserved word if, followed by an expression contained within parentheses, followed by a statement. Note that the parentheses around the expression are part of the syntax. The expression is sometimes called **a decision maker** because it decides whether to execute the statement that follows it. The expression is usually a logical expression. If the value of the expression is true, the statement executes. If the value is false, the statement does not execute and the computer goes on to the next statement in the program. The statement following the expression is sometimes called the **action statement**. Figure 2 shows the flow of execution of the if statement (one-way selection).

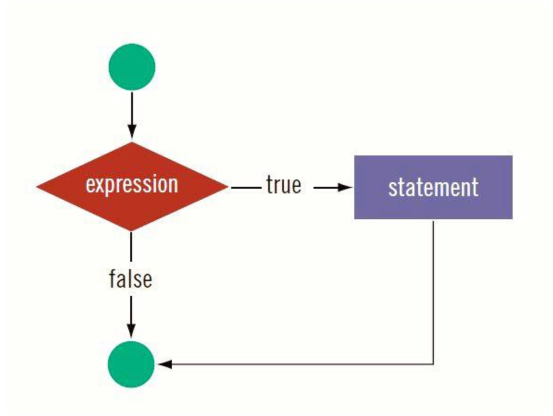


Figure 3: one way selection

Note, If there is only one statement, the curly braces may be omitted, giving the form:

```
if(condition)
      statement;
```

Example: The following C++ program finds the absolute value of an integer.

```
#include <iostream>
using namespace std;
int main() {
int number, temp;

cout << "Line 1: Enter an integer: ";
cin >> number;
cout << endl;

temp = number;
if (number < 0)
number =  - number;
cout << "The absolute value of " << temp << " is " << number << endl;
return 0;
}
```

Sample Run:
In this sample run, the user input is shaded. Enter an integer: -6734
The absolute value of -6734 is 6734

Note: Consider the following C++ statements:

```
if (score >= 60); //Line 1
grade = 'P';      //Line 2
```

Because there is a semicolon at the end of the expression (see Line 1), the if statement in Line 1 terminates. The action of this if statement is null, and the statement in Line 2 is not part of the if statement in Line 1. Hence, the statement in Line 2 executes regardless of how the if statement evaluates.

- **Two-Way Selection (if .. else)**

To implement two-way selections, C++ provides the **if. . .else** statement. Two-way selection uses the following syntax:

```
if( expression )
{
      // Execute this block if condition is true
      statementA1;
      statementA2;
      …
}else
{
      // Execute this block if condition is false
      statementB1;
      statementB2;
      …
}
```

Take a moment to examine this syntax. It begins with the reserved word if, followed by a logical expression contained within parentheses, followed by a statement, followed by the reserved word else, followed by a second statement. Statements 1 and 2 are any valid C++ statements. In a two-way selection, if the value of the expression is true, statement1 executes. If the value of the expression is false, statement2 executes. Figure 3 shows the flow of execution of the if. . .else statement (two-way selection).

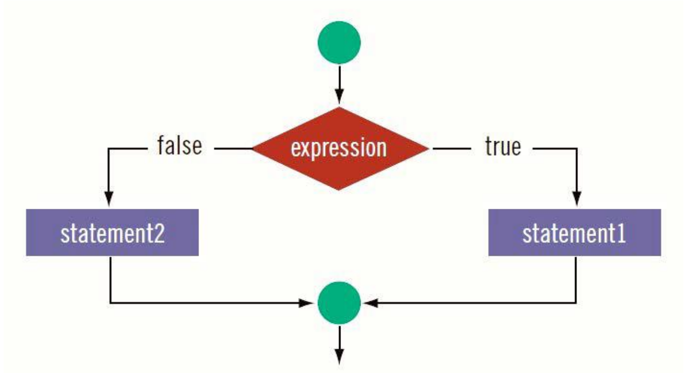


Figure 4: Two way selection

Example: The program is to check a given integer number if it is divisible by 7 or not.

```
#include <iostream>
using namespace std;
main( ){

int num;

cout<<"Please Enter The Number" <<endl ; cin>>num;

if (num % 7 == 0)
      cout << "This number is divisible by 7";
else
      cout << "This number is not divisible by 7";

return (0);
}
```

- **Multiple Selections: (If … else if)**

The *eif else if* is used to decide between two or more blocks based on *multiple* conditions:

```
if( expression )
{
      statementA1;
      statementA2;
      …
}else if ( expression )
{
      statementB1;
      statementB2;
      …
}else
{
      statementN1;
      statementN2;
}
```

If condition1 is met, the block corresponding to the if is executed. If not, then *only if* condition2 is met is the block corresponding to the else if executed. There may be more than one else if, each with its own condition. Once a block whose condition was met is executed, any else ifs after it are ignored. Therefore, in an *if-else-if* structure, either one or no block is executed.

An else may be added to the end of an if-else-if. If none of the previous conditions are met, the else block is executed. In this structure, one of the blocks *must* execute, as in a normal if-else.

Example: Here is an example using these control structures:

```
#include <iostream>
 using namespace std;

int main()
{
int x = 6;
int y = 2;

if(x > y)
      cout << "x is greater than y\n";
else if(y > x)
      cout << "y is greater than x\n";
else
      cout << "x and y are equal\n";

return (0);
}
```

The output of this program is
`x is greater than y.`

If we replace lines 5 and 6 with
Int x= 2;
Int y= 6;
then the output is
`y is greater than x.`

If we replace the lines with
Int x= 2;
Int y= 2;
then the output is
`x and y are equal.`

Nested-if
A nested if in C++ is an ***if statement*** that is the target of another ***if statement***. Nested if statements means an ***if statement*** inside another ***if statement***. **Syntax:**

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
    {
      // Executes when condition2 is true
   }
}
```

Example: C++ program to illustrate nested-if statement

```
#include <iostream>
using namespace std;

int main(){
int i = 10;

if (i == 10)
{
      // First if statement
      if (i < 15)
            cout<<"i is smaller than 15\n";

            // Will only be executed if statement above is true
            if (i < 12)
                  cout<<"i is smaller than 12 too\n";
            else
                  cout<<"i is greater than 15";
}

return 0;
}
```

Execute:

i is smaller than 15

i is smaller than 12 too

2- Selection: Switch-case

The *switch-case* is another conditional structure that may or may not execute certain statements. However, the switch-case has peculiar syntax and behavior:

```
switch(expression)
{
        case constant1:
                statementA1
                statementA2
                ...
                break;
        case constant2:
                statementB1
                statementB2
                ...
                break;
                ...
        default:
                statementZ1
                statementZ2
                ...
}
```

The `switch` evaluates `expression` and, if `expression` is equal to `constant1`, then the statements beneath `case constant 1:` are executed until a `break` is encountered. If `expression` is not equal to `constant1`, then it is compared to `constant2`. If these are equal, then the statements beneath `case constant 2:` are executed until a `break` is encountered. If not, then the same process repeats for each of the constants, in turn. If none of the constants match, then the statements beneath `default`: are executed.
Due to the peculiar behavior of `switch-case`S, curly braces are not necessary for cases where there is more than one statement (but they are necessary to enclose the entire `switch-case`). `switch-case`S generally have `if-else` equivalents but can often be a cleaner way of expressing the same behavior.

Example: Here is an example using switch-case:

```
// assignment operator
#include <iostream>
using namespace std;

int main (){
int x = 6;

switch(x) {
          case 1:
                    cout << "x is 1 \n";
                    break;
          case 2:
                    cout << "x is 2 \n";
                    break;

          case 3:
                    cout << "x is 3 \n";
                    break;
          default:
                   cout << "x is not 1, 2, or 3";
          }
}
```

Execute:

This program will print
`x is not 1, 2, or 3.`

If we replace line 5 with
`int x = 2;`
then the program will print
`x is 2.`