CST209

Object-oriented Programming C++

Prepared by Dr. Teo Bee Guan

Presented by Venantius Kumar Sevamalai

Content

Control flow in C++

- Relational operators
- Logical Operators
- Selection (if, if-else, if-else if-else, switch)
- Loops (while, for)
- BREAK and Continue Statement

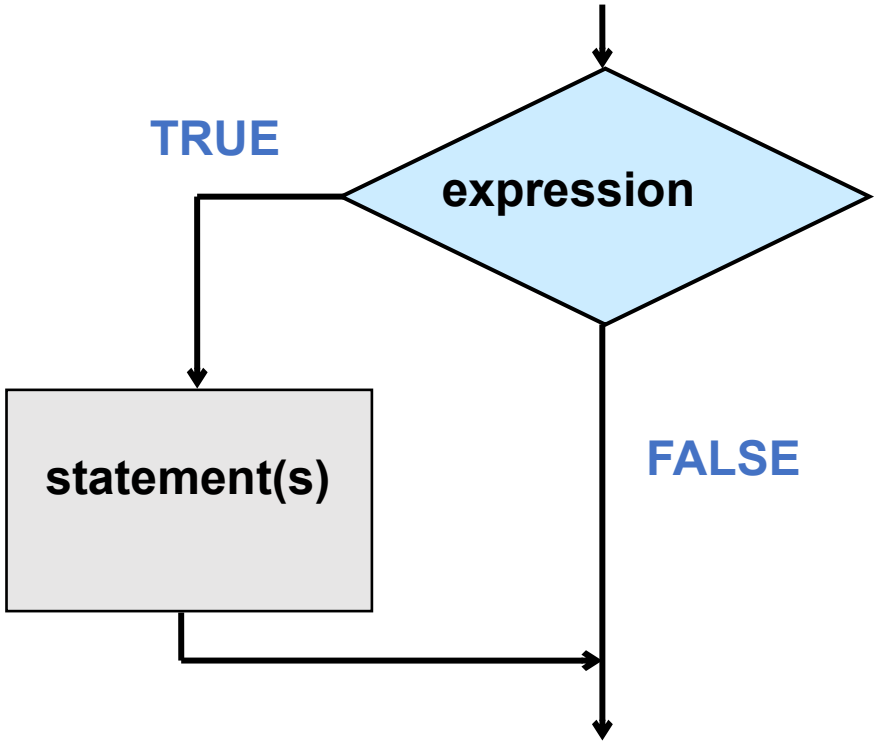
Control Flow – Relational Operator

- In C++, Relational Operators are used for comparing two or more numerical values.
- We use Relational Operators for the decision-making process.

Operator	Context
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Simple *if* Statement

- Is a **selection** of whether or not to execute a statement or a block of statement.



Simple *if* Statement Syntax

```
if  (Boolean Expression)
  Statement

if  (Bool-Expr)
{
  Statement_1
  …
  Statement_n
}
```

Practice: Code Example 1

These are NOT equivalent. Why?

- Program-1

```
if  (number == 0 )
{
    cout << “Error ” << endl;
    cout << “You entered invalid
      number.” << endl;
}
```

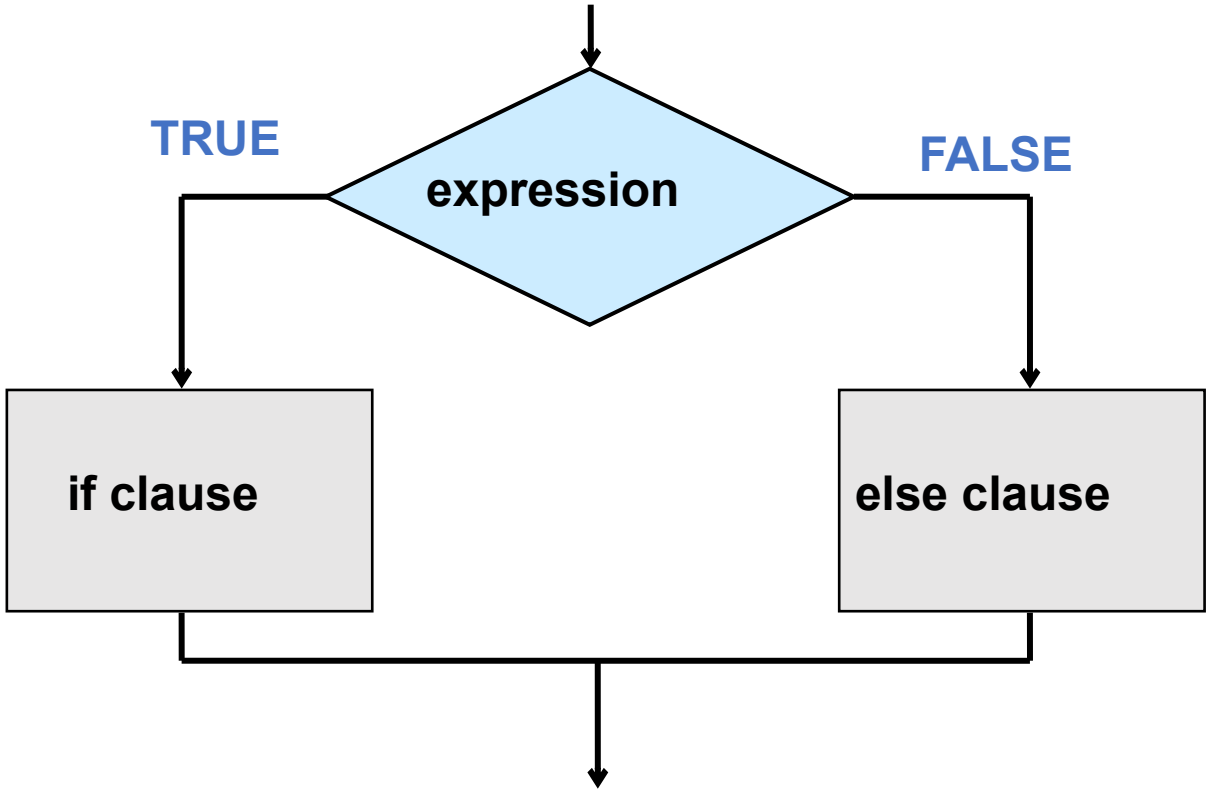
- Program-2

```
if  (number == 0 )
    cout << “Error” << endl;
    cout << “You entered invalid
    number.” << endl;
```

Practice: Code Example 2

If-else Statement

- provides **selection** between executing **one of 2 clauses** (the if clause or the else clause)



Use of blocks

- Denoted by **{ .. }**
- Recommended in controlled structures (if and loop)
- Also called compound statement.

```
if  (Bool-Expression )
{

}
else
{

}
```

"if clause"

"else clause"

Practice: Code Example 3

If-else if-else Syntax

- The if/else if statement tests a series of conditions.
- It is often simpler to test a series of conditions with the if/else if statement than with a set of nested if/else statements.

```
if (expression_1)
{
   statement
   statement
   etc.
}
```

If `expression_1` *is true these statements are executed, and the rest of the structure is ignored.*

```
else if (expression_2)
{
   statement
   statement
   etc.
}
```

Otherwise, if `expression_2` *is true these statements are executed, and the rest of the structure is ignored.*

Insert as many `else if` *clauses as necessary*

```
else
{
   statement
   statement
   etc.
}
```

These statements are executed if none of the expressions above are true.

Practice: Code Example 4

Logical Operators

Operator	Meaning	Effect
&&	AND	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
||	OR	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
!	NOT	The ! operator reverses the "truth" of an expression. It makes a true expression false, and a false expression true.

Practice: Code Example 5, 6, 7

Conditional Operators

- We can use the conditional operator to create short expressions that work like if/else statements.

Here is an example of a statement using the conditional operator:

```
x < 0 ? y = 10 : z = 20;
```

Practice: Code Example 8

The switch Statement

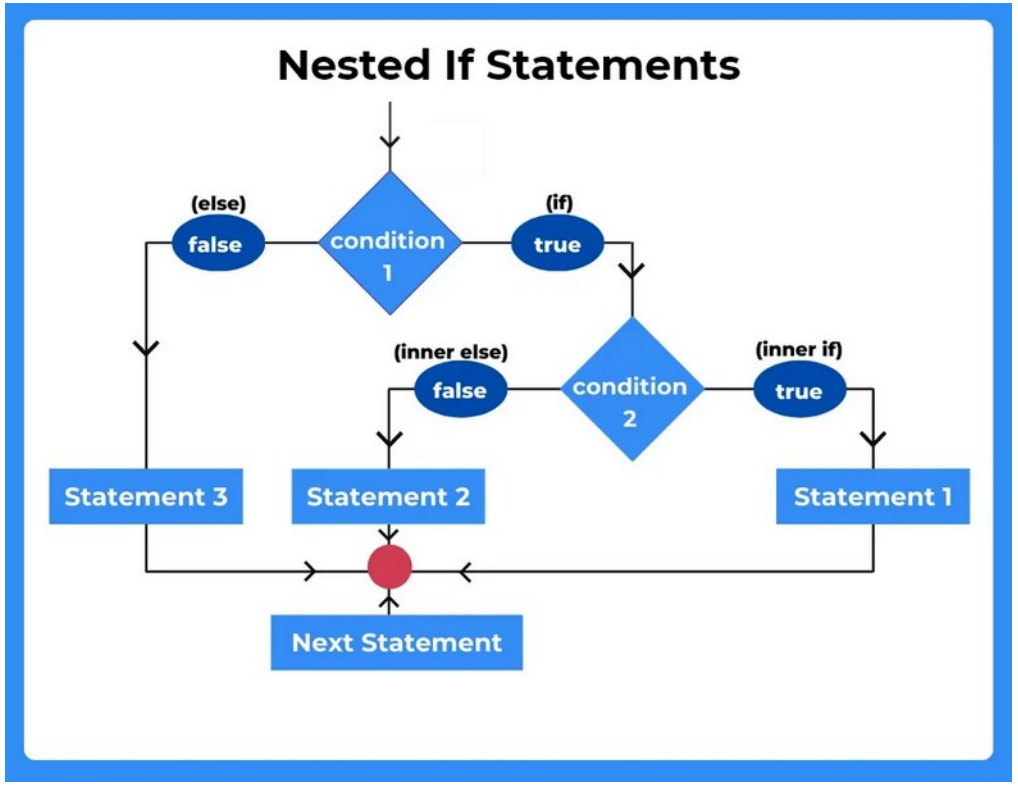
- The switch statement lets the value of a variable or expression determine where the program will branch.
- A branch occurs when one part of a program causes another part to execute.

```
switch (IntegerExpression)
{
   case ConstantExpression:
       // place one or more
       // statements here

   case ConstantExpression:
       // place one or more
       // statements here

    // case statements may be repeated as many
    // times as necessary

   default:
       // place one or more
       // statements here
}
```

Practice: Code Example 9

Nested if Statements

- To test more than one condition, an if statement can be nested inside another if statement.
- Sometimes an if statement must be nested inside another if statement.



Practice: Code Example 10

Control Flow - Loop

- Loop is a repetition control structure.
- It causes a single statement or block of statements to be executed repeatedly until a condition is met.
- There are 3 kinds of loop in C++:
  - while loop
  - do-while loop
  - for loop

While Loop

SYNTAX

```
while ( Expression )
{
        … // loop body
}
```

- No semicolon after the boolean expression
- Loop body can be a single statement, a null statement, or a block.

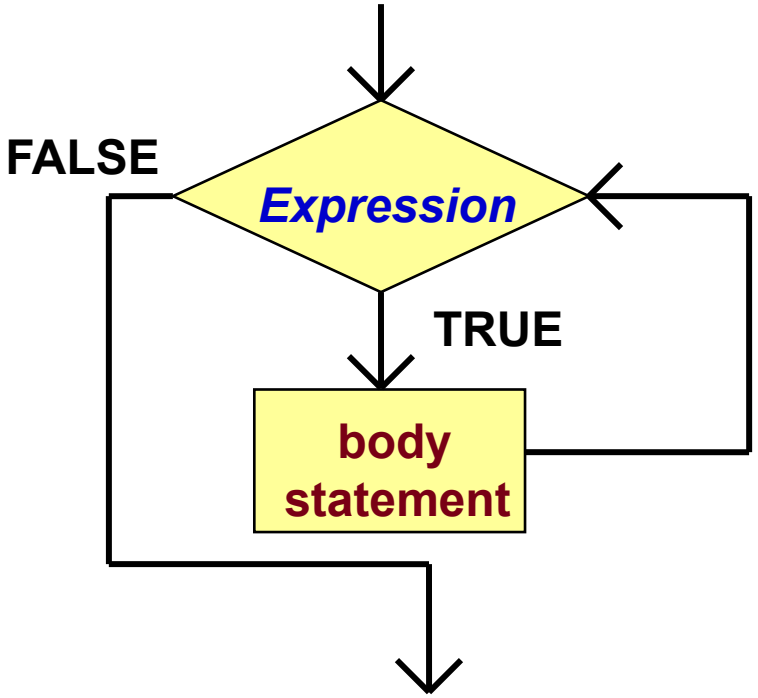
Loop Tracing

```
int   count ;
count  =  0;
while (count < 5)
{
   cout  << count  << " ";
   count =   count + 1;
}
cout  << "Done" << endl ;
```

count	Expression	Output
0	true	0
1	true	0 1
2	true	0 1 2
3	true	0 1 2 3
4	true	0 1 2 3 4
5	false	0 1 2 3 4 Done

Control Flow - Loop

- When the expression is tested and found to be false, the loop is exited and control passes to the statement which follows the loop body.



- When the expression is tested and found to be true, the loop body is executed. Then, the expression is tested again.

Practice: Code Example 11, 12

Increment and Decrement Operators

- Denoted as **++** or **--**
- Mean increase or decrease by 1
- Pre increment/decrement: ++a, --a
  - Increase/decrease by 1 before use.
- Post increment/decrement: a++, a--
  - Increase/decrease by 1 after use.
- Pre and Post increment/decrement yield different results when combining with another operation.

Do-While Loop

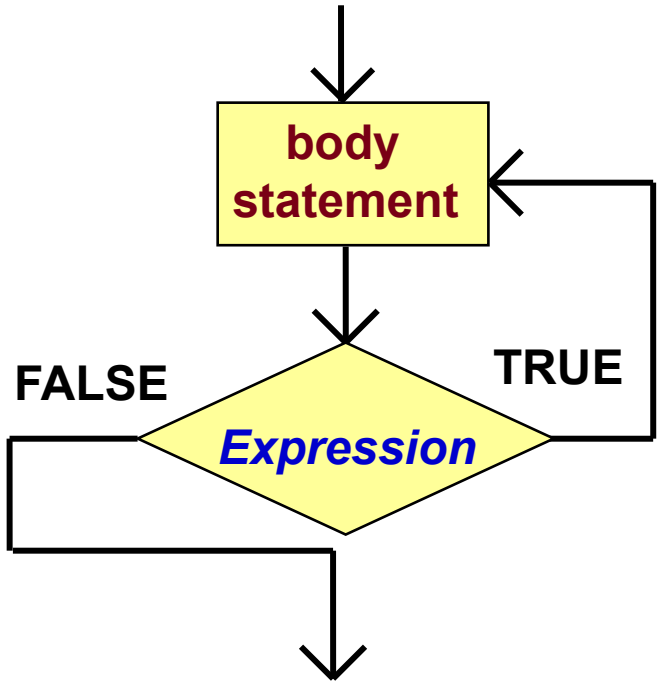
SYNTAX

```
do
{
        ... // loop body
} while ( Expression );
```

- Insured that the loop is executed at least once
- The LCV is initialized/updated before the end of the loop.
- Boolean expression is tested at the end of the loop.
- There is a semicolon after the boolean expression.

Control Flow - Loop

- The loop body is executed first



- When the expression is tested and found to be true, the loop body is executed. Then, the expression is tested again.

Practice: Code Example 13

Do-While Loop Example

```
int   ans;
do
{
   cout  << "Choose a number from 1 to 4: ";// repeated action
   cin >> ans;                            // LCV is initialized or updated
} while (ans >= 1 && ans <= 4);           // test expression
cout << "Done";
```

Output	Input	ans	Expression
Choose a number from 1 to 4:	2	2	true
Choose a number from 1 to 4:	3	3	true
Choose a number from 1 to 4:	1	1	true
Choose a number from 1 to 4:	5	5	false
Done			

for Loop

SYNTAX

```
for ( initialization; test; update )
  statement;
```

- The for loop is ideal for performing a known number of iterations.
- for loop is a count controlled loop that must have three elements:
  i. It must initialize a counter variable to a starting value.
  ii. It must test the counter variable by comparing it to a maximum value. When the counter variable reaches its maximum value, the loop terminates.
  iii. It must update the counter variable during each iteration. This is usually done by incrementing the variable.

Practice: Code Example 14, 15

See you next class