CS2311 Computer Programming

LT4: Flow Control

Conditional Statements

Outlines

- **if** statement
- Boolean logic
- **switch** statement

Syntax Summary

- Keywords

  if, else, switch, case, default

- Punctuators

  (…)

  {…}

  :

  ? :

- Operators

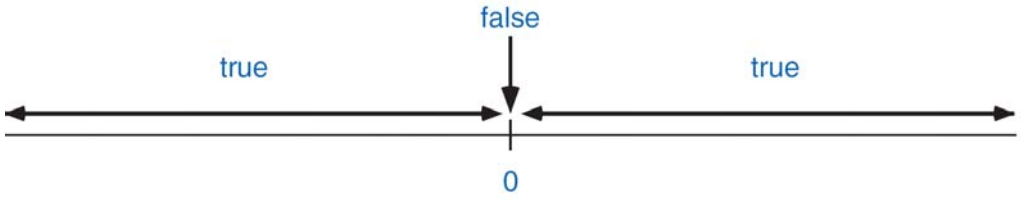
  ==, !, !=, >, >=, <, <=, &&, ||

Decisions and Actions

- In real-life:
  - We make decisions everyday
    - Go to lecture, or just hea (slack off)?
  - A decision is followed by actions
- In programming:
  - A decision is based on logical expressions
  - An action is in the form of programming statements

Logical Expressions and Operators

- Logical expressions can be true or false only
  - x == 3
  - y == x
  - x > 10
- In C++, any expression that evaluates to a non-zero value will be treated as logical true
  - 3 - 2 (true)
  - 1 - 1 (false)
  - x = 0 (false)
  - X = 1 (true)



Comparison/Relational Operators

- Binary operators which accept two operands and compare them

Relational operators	Syntax	Example
Less than	<	x < y
Greater than	>	z > 1
Less than or equal to	<=	b <= 1
Greater than or equal to	>=	c >= 2

Equality operators	Syntax	Example
Equal to	==	a == b
Not equal to	!=	b != 3

Assignment and Equality Operators

- Example:

  `x = 1`
- Assignment operator
- Place the value of the variable on the right to the variable on the left
- The value of this expression will always equal to the value on the right (1)

- Example:

  `x == 1`
- Equality operator
- T (evaluates to 1)
  - values of the value of x is 1
- F (evaluates to 0)
  - values of x is not 1
- No space between the two '='

Logical Operators

- Used to assess logical expressions
- Logical AND **&&**
  - Both conditions have to be satisfied to be true
- Logical OR ||
  - Any one condition has to be satisfied to be true
- Logical NOT **!**
  - Invert the logical result of the expression/operand

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

x	!x
true	false
false	true

Operators

Category	Examples
Arithmetic	+, -, /, *, %, =, ++, --
Comparison/relational	==, !=, >, <, >=, <=
Logical	!, &&, ||
Bitwise	~, &, |, ^, <<, >>
Compound assignment	+=, &=, <<=
Member and pointer	a[b], *, &, ->
Others	::, sizeof

Precedence & Associativity of Operators

Operator precedence (high to low)	Associativity
() ++(postfix) – (postfix)	Left to right
++(prefix) -- (prefix)	Right to left
* / %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /=	Right to left
,(comma operator)	Left to right

Conditional Statements

- In a decision making process, logical value can be used to determine what actions to take.
- Examples:
  - If it is raining, then go to AC1 canteen for lunch
  - If Mr. Trump is supporting a bill/campaign, then say bad things about it
- In programming, certain statements will only be executed when certain condition is fulfilled. We call them ***conditional statements***.

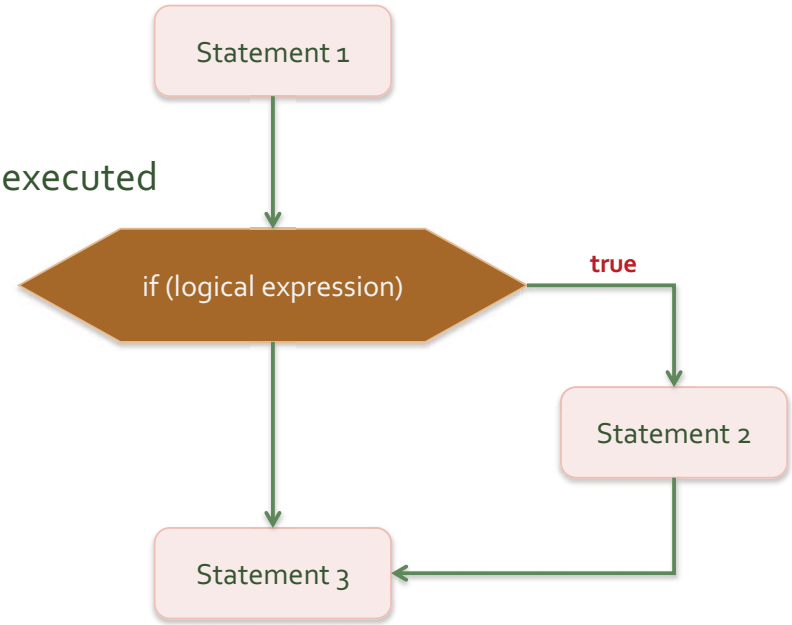
Conditional Statement : if

- One statement will be executed if the condition is true
- Syntax:

```
if (logical expression)
    statement;
```

- Only one statement will be executed
- C++ example:

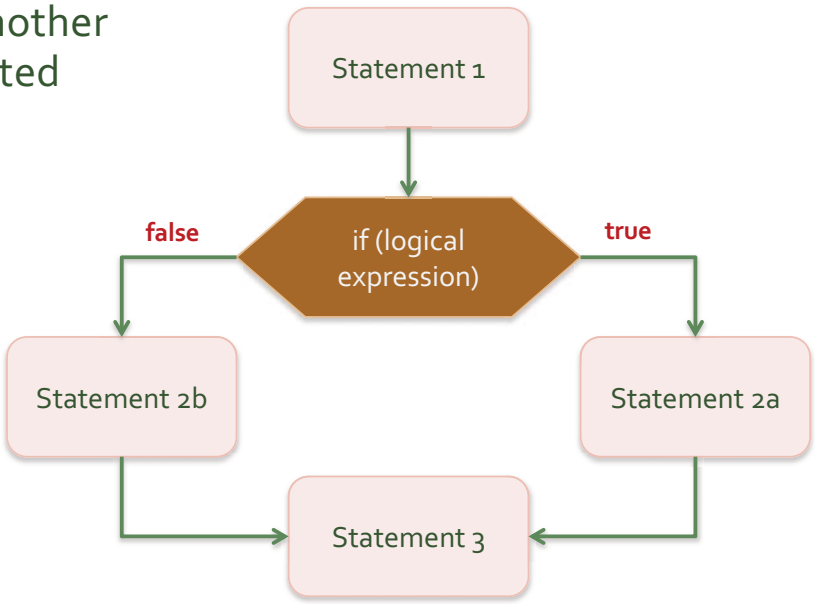
```
cin >> x;
if (x < 0)
    x = -x;
cout  << x << endl;
```



Two-way Selection

- If the condition is true, one statement will be executed. If the condition is false, another statement will be executed

```
.....
if (logical expression)
    //action for true
    statement a;
else
    //action for false
    statement b;
```



Some Points to Note

The expression should be enclosed with parenthesis ()

No semi-colon after **if** or **else**

```
if (i == 3)
    a++;
else
    a--;
```

The else part is optional

The semicolons belong to the expression statement, NOT the ***if..else*** statement

i == 3 evaluates to a non-zero value

i == 3 evaluates to zero

If i == 3 is true, increment **a** by 1, otherwise (i == 3 is false), decrement **a** by 1

Compound Statement

- Group multiple statements into one block using {} to be executed for a certain **if**, **else if**, or **else** statement
- Must use {} if there're more than one statements for an if, else, or else if statement

```
if (mark >= 90) {
        cout << "You get grade A.\n";
        cout << "Excellent!\n";
}
```


Compound Statement

We may group multiple statements to form a compound statement using a pair of braces {}

```
if (j != 3) {
        b++;
        cout << b;
}
else
        cout << j;
```

Compound statements are treated as one statement

```
if (j !=5 && d == 2) {
        j++;
        d--;
        cout << j <<d;
}
else {
        j--;
        d++;
        cout << j << d;
}
```

Multiple else-if Statements

- You can have many nested "**else if**" statements.

```
.....
if (condition 1)
        //action for true
        statement a;
else if (condition 2)
        //action for true
        statement b;
        .....
else
        //action for false
        statement;
```

An example:

```
if (score >= 90)
    grade = 'A';
else if (score >= 75)
    grade = 'B';
else if (score >= 55)
    grade = 'C';
else
    grade = 'D';
```


Beyond Two Way Condition…

- In C++, conditional statements has the following format:

```
if (logical expression 1) {
        statements when expression 1 is true
}
else if (logical expression 2) {
        statements when expression 1 is false and expression 2 is true
}
else if (…) {
        …
}
else {
        statements when all the logical expressions are false
}
```

- The ***else if*** and ***else*** part is optional
- The brackets can be omitted if the block contains one statement only

Examples – Single Statement

```
if (x > 5) {
   cout << "x is too large";
}
```

```
if (x > 5)
   cout << "x is too large";
else if (x < 3)
   cout << "x is too small";
```

```
if (x > 5)
   cout << "x is too large";
else if (x < 3)
   cout << "x is too small";
else
   cout << "x is a valid answer";
```


Examples – Compound Statements

```
if (x == 3)
   cout << "x is equal to 3";

if (x > y) {
   z = x - y;
   cout << "x is larger, the difference is " << z;
}
else {
   z = y - x;
   cout << "y is larger, the difference is " << z;
}
```

Beware of Empty Statements!

```
int x = 5;
if (x != 5);
   x = 3;
cout << x;
/*output is 3*/
```

```
int x= 5;
if (x != 5)
   x = 3;
 cout << x;
/*output is 5*/
```

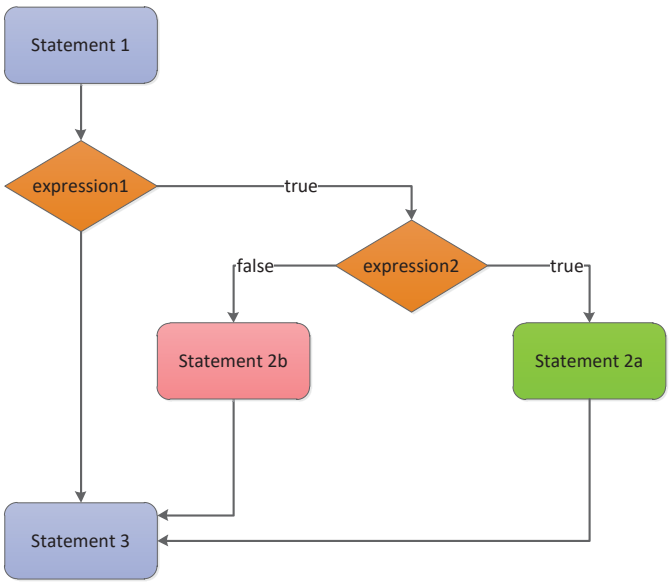
An empty statement can be specified by a semi-colon '**;**'.
Empty statement specifies that no action should be performed.

For the second program, x is assigned 3 if x is not equal to 5.

For the first program, because of the extra semi colon at the end of the if statement, nothing is executed when x != 5.

Nested if

An **if-else** statement is included with another **if-else** statement




```
statement1
if (expression1)
    if (expression2)
        statement2a
    else
        statement2b
statement3
```

Dangling **else**

- With which "**if**" the "**else**" part is associated?

```
if (a == 1)
   if (b == 2)
      cout << "***\n";
   else
      cout << "###\n";
```

```
if (a == 1)
   if (b == 2)
      cout << "***\n";
else
   cout << "###\n";
```

✗

Dangling **else** problem

- An **else** attached to the nearest **if**.

```
if (a == 1)
   if (b == 2)
      cout << "***\n";
   else
      cout << "###\n";
```

Do Not Mix == and =

```
x = 0;
y = 1;
if (x = y) {
    cout << "x and y are equal";
}
else
    cout << "unequal";
```

Output : x and y are equal

The expression x = y

- Assign the value of y to x (x becomes 1)
- The value of this expression is the value of y, i.e. 1 (which represent TRUE)
  - **FALSE** is represented by 0
  - Non-zero represents **TRUE**

Some Examples

Passing CS2311

- If you get a total mark greater than or equal to 34. You will pass the course
- Otherwise, you fail.

- Greater than or equal to is a kind of "relational operator"
  - Represented by the operator **>=** in C++
  - How to represent the above logic in C++?

Example 1a: Pass or Fail?

```
int mark;
cout << "What is your mark?\n";
cin >> mark;
if (mark >= 34)
   cout << "You passed in CS2311!\n";
```

The condition should be enclosed within **()**
If the input mark is greater than or equal to 34, the **blue** statement is executed.

Example 1b: Pass or Fail?

```
int mark;
cout << "What is your mark?\n";
cin >> mark;
if (mark >= 34) {
   cout << "You passed in CS2311! \n";
   cout << "Congratulations! \n";
}
```

If more than 1 statements are specified within an if statement, group the statements in a pair of braces **{ }**

Example 1c: Pass or Fail?

```
int mark;
cout << "What is your mark? \n";
cin >> mark;
if (mark >= 34) {
   cout << "You passed in CS2311!\ n";
   cout << "Congratulations! \n";
}
else
   cout << "You failed in CS2311… \n";
```

The **else** statement is executed when the condition **mark >= 34** is **false**

Example 1c: Pass or Fail?

```
int mark;
cout << "What is your mark? \n";
cin >> mark;
if (mark >= 34){
   cout << "You passed in CS2311! \n";
   cout << "Congratulations! \n";
}
else
   cout << "You failed in CS2311! \n";
   cout << "You should retake the course. \n";
```

Suppose the user inputs **35**. The output:

You passed in CS2311!
Congratulations!
You should retake the course.

Why?

Example 1c: Pass or Fail?

```
int mark;
cout << "What is your mark? \n";
cin >> mark;
if (mark >= 34) {
   cout << "You passed in CS2311! \n";
   cout << "Congratulations! \n";
}
else {
   cout << "You failed in CS2311! \n";
   cout << "You should retake the course. \n";
}
```

Include a **brace** to group the statements in the **else** part

Example 2a: Mark To Grade Conversion

```
if (mark >= 70 && mark <= 100)
    cout << "A";
if (mark >= 55 && mark < 70)
    cout << "B";
if (mark >= 45 && mark < 55)
    cout << "C";
if (mark >= 34 && mark < 45)
    cout << "D";
if (mark < 34 && mark > 0)
    cout << "F";
if (mark < 0 || mark > 100)
    cout << "Invalid Grade";
```


Mark to Grade Conversion (**else-if** version)

```
if (mark < 0 || mark > 100)
   cout << "Invalid Grade";
else if (mark >= 70)
   cout << "A";
else if (mark >= 55)
   cout << "B";
else if (mark >= 45)
   cout << "C";
else if (mark >= 34)
   cout << "D";
else
   cout << "F";
```

The "**else if**"or "**else**"part is executed only if all the preceding conditions are **false**

C++ Syntax is Different from the Math Syntax

```
if (mark >= 70 && mark <= 100)
    ……
```

Can we express the above condition as follows?

```
if (70 <= mark <= 100)
    ……
```

Ans: NO

Nested-if

- An if statement can be nested within another if statement

```
if (mark >= 70 && mark <= 100) {
  if (mark > 90)
      cout << "You get grade A+.\n";
  else if (mark > 80)
      cout << "You get grade A.\n";
  else
      cout << "You get grade A-.\n";
}
else if (…)
  ……
```

Example 2b

- Modify the previous program such that if the mark is 100, the statement "**Full mark!**" should be printed (in addition to the grade).

The Program

```
if (mark >= 70 && mark <= 100) {
   if (mark > 90) {
        cout << "You get grade A+.\n";
        if (mark == 100)
                cout << "Full mark! \n";
   }
   else if (mark > 80)
        cout << "You get grade A. \n";
   else
        cout << "You get grade A-. \n";
}
else if ……
```

Short-circuit Evaluation

- Evaluation of expressions containing **&&** and **||** stops as soon as the outcome is known.
  - This is called ***short-circuit evaluation***
- Short-circuit evaluation improves program efficiency
- Short-circuit evaluation exists in some other programming languages too, e.g., C and Java

Short-circuit Evaluation

- Given integer variables i, j and k, what are the outputs when running the program fragment below?

```
k = (i=2) && (j=2);
cout << i << j << endl; /* 2  2 */
k = (i=0) && (j=3);
cout << i << j << endl; /* 0  2 */
k = i || (j=4);
cout << i << j << endl; /* 0  4 */
k = (i=2) || (j=5);
cout << i << j << endl; /* 2  4 */
```

switch Statement

```
switch (expression) {
   case constant-expr1 : statement1
   case constant-expr2 : statement2
   …
   …
   case constant-exprN : statementN
   default : statement
}
```


Example

```
int x;
cin >> x;
switch (x) {
    case 0:
        cout << "Zero";
        break; /* no braces is needed */
    case 1:
        cout << "One";
        break;
    case 2:
        cout << "Two";
        break;
    default:
        cout << "Greater than two";
} //end switch
```

switch Statement

- Semantics
  - Evaluate the ***switch expression*** which results in an integer type (**int, long, short, char**)
  - Go to the ***case*** label having a constant value that matches the value of the **switch** expression; if a match is not found, go to the ***default*** label; if ***default*** label does not exist, terminate the **switch**
  - Terminate the **switch** when a ***break*** statement is encountered
  - If there is no ***break*** statement, execution "***falls through***" to the next statement in the successful case

A Segment Using switch

```
int main() {
    short odd_num = 0, even_num = 0, other_char = 0;
    char c;

    while ((c = getchar()) != '.') {  /* get a char */
        switch (c) {
            case '1': case '3': case '5': case '7': case '9':
              odd_num++;
              break;
            case '0': case '2': case '4': case '6': case '8':
              even_num++;
              break;
            default:
              other_char++;
              break;
        }
    }
    cout << "Number of even numbers is " << even_num << endl;
    return 0;
}
```

Ternary Conditional Operator (?:)

- Syntax:

  expr1 ? expr2 : expr3

- Semantics
  - **expr1** is evaluated
  - If the result is true, execute **expr2** ; otherwise **expr3** is executed
  - The value of the whole expression is the value of expression evaluated at the end
  - Data type of the returning value is determined by both **expr2** and **expr3**, but not the one being evaluated ultimately
- Example

  int min_x = (x > y) ? y : x;

NOT recommended

Summary

- Boolean logic has two values only; **true** or **false**.
- Conditional statements are the statements that only execute under certain conditions.
- There are two types of conditional statements

  if (…) {…} else {…}

  switch(…) {…case : break}