Simple Control Flow: `if`-`else` Statements

Lecture 4

Outline

1. Branching
2. Boolean Expressions
3. Sanitizing Inputs

Control Flow

- Control flow is the order in which program statements are executed
- So far, all of our programs have been executed straight-through from the first statement to the last
- In general, you will need more complex control flow
- For example, to choose between two (or more) possibilities

Branching

“Human” example: greeting two different people

Basic Steps

- Thought process
  - Whom am I looking at?
  - If I am looking at the Special
    - Say "Hi Emmet"
  - If I am looking at Wyldstyle
    - Say "Hi Lucy"

- C++ uses **`if`-`else`** statements to choose between alternatives

if – else (1)

Example:

```
if ( user == “Special” )
{
    cout << “Hi Emmet” << endl;
}
else
{
    cout << “Hi Lucy” << endl;
}
```

Generic form:

```
if ( BOOLEAN EXPRESSION )
{
    YES/TRUE STATEMENT(S)
}
else
{
    NO/FASE STATEMENT(S)
}
```

`if` – `else` (2)

- Each `if`-`else` statement allows your program to do one of two different things
- In other words, you EITHER use one set of statements OR you use the other
- The statements in between the `{ }` for the `if` and the `else` can contain any code you want, just like code that is not inside an `if`-`else`
  - Even other `if`-`else` statements!

Boolean Expressions

Boolean expressions (like `bool` variables) are either true or false and are composed of comparisons

Comparison Operators

`==`	Equal To	`if ( x == 10 )`
`!=`	Not Equal To	`if ( day != “Sunday” )`
`<`	Less Than	`if ( hours < 40 )`
`<=`	Less Than or Equal To	`if ( dollars <= 100 )`
`>`	Greater Than	`if ( dollars > 100 )`
`>=`	Greater Than or Equal To	`if ( hours >= 40 )`

Example

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

int main()
{
	double input;
	cout << "Please enter a number: ";
	cin >> input;

	if ( input > 100 )
	{
		cout << "That is greater than 100." << endl;
	}
	else
	{
		cout << "That is less than or equal to 100." << endl;
	}

	cout << "Thank you!" << endl;

	return 0;
}
```

Control Flow

- So, if the `if` condition is <u>true</u>, the statements between the next curly braces are executed, then it skips down past the `else` curly braces and continues executing

- If the `if` condition is <u>false</u>, the statements between the `if` braces are ignored, then the statements between the `else` braces are executed, then it continues executing whatever is next

Example

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

int main()
{
     int x = 15;

     if ( x > 0 )
     {
          cout << ”x is positive." << endl;
     }
     else
     {
          cout << ”x is non-positive." << endl;
     }

     cout << "Thank you, and good night." << endl;

     return 0;
}
```

15 > 0 is true

Exercise

Write a program that reads an integer from the user and determines whether or not the integer is even (evenly divisible by 2) or odd (not evenly divisible by 2).

Answer

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

int main()
{
	int input, remainder;

	cout << "Please enter a number: ";
	cin >> input;

	remainder = input % 2;

	if ( remainder == 0 )
	{
		cout << input << ” is even." << endl;
	}
	else
	{
		cout << input << ” is odd." << endl;
	}
	return 0;
}
```

Omitting Braces

- You can omit the curly braces after an `if` or `else` ONLY if there is exactly one statement
- Example:

```
if ( hours_worked > 8 )
   cout << “Quitting time!” << endl;
else
   cout << ”Get back to work!" << endl;
```

- I recommend always using the braces
  – Easier to read
  – Less chance of errors if you add statements later

Multiple Choices

- Standard `if`-`else` statements give you two choices
- You can have more choices by adding `else if` statements in between the `if` and `else`
- Any number of `else if` statements can be added after an `if` statement
- Each will be tested in order until one of the conditions is true, and the `else` statements will only run if none of the conditions are true
- You can always omit the `else` statement if want (even if you have no `else if` statements)

else if Statements

```
if ( EXPRESSION1 )
{
   // run statements if expression 1 is true
}
else if ( EXPRESSION2 )
{
   // run statements if expressions 1 is false,
   // and expression 2 is true
}
else
{
   // run statements if both expressions are false
}
```

Example

```
if ( x > 10 )
{
   cout << "x is greater than 10" << endl;
}
else if ( x < 5 )
{
   cout << "x is less than 5" << endl;
}
else
{
   cout << "x is between 5 and 10" << endl;
}
```

Multiple `else if` Statements

```
if ( x > 10 )
{
    cout << "x is greater than 10" << endl;
}
else if ( x > 5 )
{
    cout << "x is between 5 and 10" << endl;
}
else if ( x > 0 )
{
    cout << "x is between 0 and 5" << endl;
}
else
{
    cout << "x is less than 0" << endl;
}
```

Exercise

Write a program that reads a numeric score from the user and outputs a letter grade

- If the score is >=90, output A
- If the score is >=80 and <90, output B
- If the score is >=70 and <80, output C
- If the score is >=60 and <70, output D
- If the score is <60, output F

Answer

```
int score;
cout << “Enter your score: “;
cin >> score;

cout << endl << “Letter grade: “;
if ( score >= 90 )
{
      cout << “A” << endl;
}
else if ( score >= 80 )
{
      cout << “B” << endl;
}
else if ( score >= 70 )
{
      cout << “C” << endl;
}
else if ( score >= 60 )
{
      cout << “D” << endl;
}
else
{
      cout << “F” << endl;
}
```

Complex Boolean Expressions

Multiple comparisons can be combined with the "and" and "or" operators

&& is the "and" operator

- Example: `( ( hours > 20 ) && ( hours <= 40 ) )`
- Entire expression is true only if *both* comparisons are true, and false if *either* is false

|| is the "or" operator

- Example: `( ( days == 30 ) || ( days == 31 ) )`
- Entire expression is true if *either* comparison is true, and false only if *both* are false

Examples

```
// assume x = 5                // false, (5 >= 10) is false, so it
(x >= 10) && (x != 12)            doesn't matter if (x != 12) is true

// assume x = 48               // true, (48 >= 10) is true and
(x >= 10) && (x != 12)            (48 != 12) is true

// assume x = 12               // false, (12 >= 10) is true but
(x >= 10) && (x != 12)            (12 != 12) is false

// assume x = 12               // true, (12 >= 10) is true, so it
(x >= 10) || (x != 12)            doesn't matter if (x != 12) is true

// assume x = 5                // true, (5 >= 10) is false, but
(x >= 10) || (x != 12)            (5 != 12) is true
```

Example: Time

```
int hours;
cout << "Enter the hours in 24 hour notation: ";
cin >> hours;

if ( ( hours >= 1 ) && ( hours <= 11 ) )
{
      cout << "That is " << hours << "am" << endl;
}
else if ( ( hours >= 13 ) && ( hours <= 23 ) )
{
      hours = hours - 12;
      cout << "That is " << hours << "pm" << endl;
}
else if ( hours == 12 )
{
      cout << "That is " << hours << "pm" << endl;
}
else if ( hours == 0 )
{
      cout << "That is 12am" << endl;
}
else
{
      cout << "That is not a valid hour!" << endl;
}
```

Example: Pinhole Camera

```
double focal_length; // f
double object_distance; // o
double image_distance; // i
double step1;

cout << "Enter the focal length: ";
cin >> focal_length;
cout << "Enter the object distance: ";
cin >> object_distance;

if ( ( focal_length == 0 ) || ( object_distance == 0 ) )
{
       cout << "Please enter non-zero values." << endl;
       return 0;
}

// solve the thin lens equation: 1/i = 1/f - 1/o
step1 = ( 1 / focal_length ) - ( 1 / object_distance );
if( step1 == 0 )
{
       cout << "The image is infinitely far away." << endl;
}
else
{
       image_distance = 1 / step1;
       cout << "The image distance is " << image_distance << "." << endl;
}
```

Sanitizing Inputs

- When you get input from the user, you should always check that the values they entered are reasonable
  - Otherwise you might have a GIGO scenario: Garbage In, Garbage Out
  - That is, if they input "bad" values into you program, they will get "bad" results

- You can stop your program early by adding additional `return` statements
  - Typically after you detect an error or bad input value

Exercise

Write a program that reads two exam scores from the user. If both exam scores are greater than 90, print "Way to go!". If either exam score is less than 70, print "Study more!". Otherwise, print "Keep it up!".

Answer

```
int exam1;
int exam2;

cout << "Enter your two exam scores:" << endl;
cin >> exam1 >> exam2;

if ( ( exam1 > 90 ) && ( exam2 > 90 ) )
{
    cout << "Way to go!" << endl;
}
else if ( ( exam1 < 70 ) || ( exam2 < 70 ) )
{
    cout << "Study more!" << endl;
}
else
{
    cout << "Keep it up!" << endl;
}
```

Common Mistakes

- When using comparison operators, always use == for checking equality, never =
  - Most compilers will not give you any errors or warnings, but your program will not operate as you expect

- You can not put multiple comparisons together with the and/or operators
  - Incorrect: `if ( 10 < x < 15 )`
  - Correct: `if ( ( x > 10 ) && ( x < 15 ) )`

Wrap Up

- Use the `if-else` statement when you need to choose between two possibilities
- Always put Boolean expressions in parentheses
- Multiple expressions can be put together with the "and" (**&&**) and "or" (**||**) operators
  - Always put each comparison in its own set of parentheses
- Always use == for testing equality, never =