CS31 INTRODUCTION TO COMPUTER SCIENCE

Fall 2015

Discussion Session 2

Noor Abani

VARIABLES

- We might need to give a certain value a name so we can refer to it later. We do this using variables.
- Variables can be of a variety of types, whether it's an integer number (no decimal), a decimal number, a string of characters, etc…
- Variable Declaration:
  - syntax: <type> <name>;
  - Optionally, we can assign a value for our variable when we declare it
- The name of a variable is called an identifier
  - A C++ identifier must start with either a letter or the underscore symbol,
  - All the rest of the characters must be letters, digits, or the underscore symbol

  Examples of valid identifiers: x, y2, _speed, COUNT

  Examples of invalid identifiers: 3, 3z, %cnt, rate-1
- Note: C++ is case sensitive i.e. Count, count and COUNT are three different identifiers that can be used to name three different variables.

EXAMPLES

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

int main(){

    int x;

    int theFirstNum,
    theSecondNum;

    double testAssignment = 5.01,
    variableNameThisIsSoLong = -10.0;

    int 2legit2int = 2;

}
```

Declaring a variable of type int

We can declare multiple variables at once

We can assign values to variable when declaring them

This will not compile: Variables must consist of alphanumeric values plus underscores and CANNOT begin with a number.

EXAMPLES

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

int main(){

        int x;

        cout << x << endl;
}
```

When a primitive variable is not initialized (i.e., set to some value like int x = 5;), it will have unpredictable junk value.

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

int main(){

        int x = 0.5;

        cout << x << endl;
}
```

When a value that is incompatible with a variable type is attempted to be placed in that variable, type coercion may be well defined. This is when a value of type A has behavior that allows it to be stored in variable of type B. In this example, when a double is attempted to be placed in an int, we shave off the decimal.

EXAMPLES

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

int main(){
 int numerator = 5,
        divisor = 2,
        result = numerator / divisor;
        cout << result << endl;
       }
```

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

int main(){

int numerator = 5,
        divisor = 2;
        double result = numerator / divisor;
        cout << result << endl;
}
```

Both these programs will print out 2.

To get the correct answer, one of our variables needs to be declared as a double

SIMPLE C++ DATA TYPES

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	–32,768 to 32,767	Not applicable
`int`	4 bytes	–2,147,483,648 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	–2,147,483,648 to 2,147,483,647	Not applicable
`float`	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
`double`	8 bytes	approximately 10^{-308} to 10^{308}	15 digits
`long double`	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
`char`	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
`bool`	1 byte	`true`, `false`	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

ASSIGNMENT STATEMENTS

- The most direct way to change the value of a variable
- In C++, the equal sign is used for assignment
- Always consists of a variable on the left-hand side of the equal sign and an expression on the right-hand side
  - Examples: salary = rate * numOfHours;

    temperature = 98;

    i = i + 2;

MORE ASSIGNMENT STATEMENTS

- A shorthand notation exists that combines the assignment operator (=) and an arithmetic operator so that a given variable can have its value changed by adding, subtracting, multiplying by, or dividing by a specified value.

EXAMPLE	EQUIVALENT TO
`count += 2;`	`count = count + 2;`
`total -= discount;`	`total = total - discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time / rushFactor;`
`change %= 100;`	`change = change % 100;`
`amount *= cnt1 + cnt2;`	`amount = amount * (cnt1 + cnt2);`

CONSTANTS

- When you initialize a variable inside a declaration, you can mark the variable so that the program is not allowed to change its value. To do this, place the word **const** in front of the declaration, as described here:
  - **SYNTAX:**

    const *Type_Name Variable_Name = Constant*;

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

int main(){

    const  int numberOfBranches = 10;
    numberOfBranches = 16;
}
```

This program will not compile.

OPERATORS-1

- **Operators** are symbolic keywords used to compute values between operands, or the arguments to the operators.
- **Unary vs. Binary Operators**:
  - Unary operators operate on a single argument value
  - whereas binary operators operate on two. Binary operators have an lvalue and an rvalue, on the left and right sides of the operator, respectively.
- Some binary operators we have seen:

- **Assignment (=):** Assigns the rvalue to the variable designated by the lvalue.
- **Addition (+):** Returns the sum of the lvalue and the rvalue.
- **Subtraction (-):** Returns the difference of the lvalue and the rvalue.
- **Multiplication (*):** Returns the product of the lvalue and the rvalue.
- **Insertion (<<):** Inserts the data in the rvalue into the stream in the lvalue.
- **Extraction (>>):** Extracts the data from stream in the lvalue into the target in the rvalue.
- **Scope Access (::):** Returns the entity named by the rvalue in the scope named by the lvalue.

OPERATORS-2

- Some unary operators

  - **Negation (-):** Returns the negative of the rvalue.
  - **Increment (++):** Increments either the lvalue or rvalue (depending on which side of a variable the ++ is on) by 1.
  - **Decrement (--):** Decrements either the lvalue or rvalue (depending on which side of a variable the ++ is on) by 1.

- Postfix increment / decrement (x++;) returns the current value of the variable before incrementing.
- Prefix increment / decrement (++x;) increments the current value of the variable before returning.

EXAMPLES

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

int main(){
        int n = 2;
        int valueProduced = 2*(++n);
        cout << valueProduced << "\n";
        cout << n << "\n";
        }
```

Output: 6
3

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

int main(){
        int n = 8;
        int valueProduced = n- -;
        cout << valueProduced << "\n";
        cout << n << "\n";
}
```

Output: 8
7

PRECEDENCE RULES

- What is precedence?
  - The order in which operators evaluate their operands. Precedence is a property of each operator.
  - **Evaluate lower precedence operators first, and in the event of a precedence level tie, operate from left to right.**
  - You can specify the order of operations in an arithmetic expression by inserting parentheses.

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right

2	`++ --`
	type`()` *type*`{}`
	`()`
	`[]`
	`.`
	`->`
		Suffix/postfix increment and decrement
		Function-style type cast
		Function call
		Array subscripting
		Element selection by reference
		Element selection through pointer	

3	`++ --`
	`+ -`
	`! ~`
	`(`*type*`)`
	`*`
	`&`
	`sizeof`
	`new, new[]`
	`delete, delete[]`
		Prefix increment and decrement
		Unary plus and minus
		Logical NOT and bitwise NOT
		C-style type cast
		Indirection (dereference)
		Address-of
		Size-of[note 1]
		Dynamic memory allocation
		Dynamic memory deallocation
			Right-to-left

4	`.* ->*`	Pointer to member	Left-to-right
5	`* / %`	Multiplication, division, and remainder	
6	`+ -`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=` `> >=`	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
9	`== !=`	For relational = and ≠ respectively	
10	`&`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	

15	`?:`
	`=`
	`+= -=`
	`*= /= %=`
	`<<= >>=`
	`&= ^= |=`
		Ternary conditional[note 2]
		Direct assignment (provided by default for C++ classes)
		Assignment by sum and difference
		Assignment by product, quotient, and remainder
		Assignment by bitwise left shift and right shift
		Assignment by bitwise AND, XOR, and OR
			Right-to-left

16	`throw`	Throw operator (for exceptions)	
17	`,`	Comma	Left-to-right

EXAMPLES

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

int main(){
        int a = 5,
              b = a = a + 5 * 2;
            cout << a << endl;
            cout << b << endl;
        }
```

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

int main(){
        int x = -3,
         y = 5,
         z = 8 * (x = y - 4);

      cout << x << endl;
      cout << z << endl;
}
```

precedence levels: (=, 15), (+, -, 6), (*, 5)

```
Output: 15
            15
```

```
Output: 1
              8
```

STRINGS

- **Strings** are objects that represent a sequence of characters (in other words: a sequence of text)
- Unlike types int and double, which are native to the C++ language, strings are a std library feature class.
- **String Concatenation:** The process by which we add strings to each other, end to end. To do so, we simply use a +.
- **String Length:** We can determine how many characters are in a string by using its .length() method.
- **Character Access:** We can retrieve the ith character of a string via the syntax: stringName[i]

EXAMPLES

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

int main(){
        string me = "Noor";
        string space = " ";
        string result = me + "is an awesome TA";
        cout << result << endl;
        string betterResult = me + space + "is an awesome
        TA";
        cout << betterResult <<endl;
        betterResult += betterResult;
        cout << betterResult << endl;
        }
```

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

int main(){
      string test = "hi";
      int five = 5;

      cout << test << five << endl;

     // what if I do this:
     // test = test + five;
     // cout << test << endl;
}
```

EXAMPLES

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

int main() {
string me = "Noor",
blank = "",
space = " ",
nLine = "\n",
result;

cout << me << ": " << me.length() << endl;
cout << "blank" << ": " << blank.length() << endl;
cout << "space" << ": " << space.length() << endl;
cout << "newLine" << ": " << nLine.length() << endl;
cout << "result" << ": " << result.length() << endl;
}
```

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

int main() {
string str = "Awesome";
cout << str[0] << endl;
cout << str[1] << endl;
cout << str[6] << endl;
//cout << str[8] << endl;
}
```

CONSOLE OUTPUT

- Output Using cout
- Any combination of variables and strings can be output.

```
cout << "Hello!\n"
<< "Welcome to C31.\n";
```

- you can output any number of items, each either a string, a variable, or a more complicated expression.
- Simply insert a << before each thing to be output.

Ex:
```
cout << numberOfGames << " games played.";
```

```
cout << numberOfGames;
cout << " games played.";
```

MORE ON COUT

- You can include arithmetic expressions in a cout statement

```
cout << "The total cost is $" << (price + tax);
```

- The computer does not insert any extra space before or after the items output by a cout statement

```
cout << firstNumber << " " << secondNumber;
```

- If you wish to insert a blank line in the output, you can output the newline character
  - \n by itself: cout << “\n”;
  - endl: cout << endl;

FORMATTING FOR NUMBERS WITH A DECIMAL POINT

- There is a “magic formula” that you can insert in your program to cause numbers that contain a decimal point, such as numbers of type double , to be output with the exact number of digits after the decimal point that you specify.
  - cout.setf(ios::fixed);
  - cout.setf(ios::showpoint);
  - cout.precision(n);

tells the cout stream to format numbers with fixed number of digits.

Display a decimal and extra zeros, even when not needed.

tells the cout stream to only display n digits and round appropriately (if in default formatting), or the number of digits past the decimal if in fixed.

EXAMPLE

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

int main() {
double price = 78;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout << price << endl;

double anotherdbl = 65.2183145;
cout << anotherdbl << endl;

cout.precision(5);
cout << anotherdbl << endl;
}
```

CONSOLE INPUT-1

- Input from the user with the same way you use cout for output.
- The syntax is similar, except that cin is used in place of cout and the arrows point in the opposite direction.

```
cin >> numberOfLanguages;
```

- When a program reaches a cin statement, it waits for input to be entered from the keyboard. It sets the first variable equal to the first value typed at the keyboard, the second variable equal to the second value typed, and so forth.
- Numbers in the input must be separated by one or more spaces or by a line break. These delimiting characters are called **whitespace**
- When you use cin statements, the computer will skip over any number of blanks or line breaks until it finds the next input value. Thus, it does not matter whether input numbers are separated by one space or several spaces or even a line break.

CONSOLE INPUT-2

- What if the string to be inputted contains spaces?
  - Use **getline** instead

    Syntax: string s;
    getline(cin, s);

  - **Getline** command consumes all characters up to, and including, the new line character. It then throws away the new-line, and stores the resulting string in s.

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

int main() {
string dogname;

cout << "What is your dog's name?" << endl;
cin >> dogname;
cout << "The name is: " << dogname << endl;
// Try input Rex
//Try input Mr. Bonjo
}
```

ISSUE!

- Since cin >> var; leaves a newline, and getline(cin, s) consumes all characters up to and including a new line, using the first before the second will cause the leftover newline character to be interpretted as hitting enter without any text on the second.
- How to Fix this?
  - **cin.ignore(n, pattern)** ignores n characters or until the first encountered instance of pattern from the input stream.
- NOTE: We ONLY use cin.ignore(n, pattern) when we've used cin and then directly after use getline!

The character to be ignored. In this case it is the newline character i.e. '\n'

EXAMPLE

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

 int main () {
    string inputString;
    int inputInt;
    cout << "Enter a number: ";
    cin >> inputInt;
    cout << "Input was: "
        << inputInt << endl;
cout << "Enter a string: ";
    getline(cin, inputString);
    cout << "Input was: "
        << inputString << endl;
 }
```

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

 int main () {
    string inputString;
    int inputInt;
    cout << "Enter a number: ";
    cin >> inputInt;
    cout << "Input was: "
        << inputInt << endl;
    cin.ignore(1, '\n');
    cout << "Enter a string: ";
    getline(cin, inputString);
    cout << "Input was: "
        << inputString << endl;
 }
```

BOOLEAN EXPRESSIONS

- Reminder:
  - Data type bool. It takes values true or false.
  - Values of type bool can be assigned to variables of an integer type (short , int , long), and integers can be assigned to variables of type bool.
    - any **nonzero** integer will be stored as the value true . Zero will be stored as the value false
    - When assigning a bool value to an integer variable, true will be stored as 1 , and false will be stored as 0 .
- A **Boolean expression** is any expression that is either true or false.`

BUILDING BOOLEAN EXPRESSIONS

- Comparison operators:

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	==	x + 7 == 2*y	x + 7 = 2y
≠	Not equal to	!=	ans != 'n'	*ans* ≠ *'n'*
<	Less than	<	count < m + 3	*count* < *m* + 3
≤	Less than or equal to	<=	time <= limit	*time* ≤ *limit*
>	Greater than	>	time > limit	*time* > *limit*
≥	Greater than or equal to	>=	age >= 21	*age* ≥ 21

COMBINING BOOLEAN EXPRESSIONS

- We can evaluate multiple multiple Boolean expressions using the and (&&) and or (||) operators.

AND

Exp_1	*Exp_2*	*Exp_1 && Exp_2*
true	true	true
true	false	false
false	true	false
false	false	false

OR

Exp_1	*Exp_2*	*Exp_1 || Exp_2*
true	true	true
true	false	true
false	true	true
false	false	false

NOT

Exp	*!(Exp)*
true	false
false	true

EXAMPLES

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

int main() {
int count = 0;
int limit = 10;
cout << ((count == 0) && (limit < 20)) << endl;
cout << (count == 0 && limit < 20) << endl;
cout << ((limit > 20) || (count < 5)) << endl;
cout << (!(count == 12)) << endl;
cout << ((5 && 7) + !16) << endl;
}
```

FLOW CONTROL

- If-statement:

```
If (Boolean Expression)
        statement;
```

- If-else statement:

```
if (Boolean Expression) {
    // Execute everything in here if
    // conditional was evaluated to true
  } else {
    // Execute everything in here if
    // Boolean exp. was evaluated to false
  }
```

FLOW CONTROL -2

- Chaining if-statements

```
if (Boolean Expression 1) {
   // Execute everything in here if
   // conditional1 was evaluated to true
  } else if (Boolean Expression 2) {
   // Execute everything in here if
   // conditional1 was evaluated to false AND
   // conditional2 was evaluated to true
  } else {
   // Execute everything in here if
   // conditional1 was evaluated to false AND
   // conditional2 was evaluated to false
  }
```

EXAMPLES

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

int main() {
int a = 5;

if (a > 5) {
        cout << "It is greater than 5";
        }
        else {
        cout << "no its not";
        }
        }
```

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

int main() {
string greeting= “hey";

if (greeting == "hello") {
cout << “Hello to you.\n";
return 0;
}
cout << "Anyone here?\n";
}
```

Note that the curly brackets are not mandatory but if not used only the first statement after the *if* will be executed. It is better to always include the curly brackets.

EXAMPLES

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

int main() {
string greeting = "hello";
int x = 4,
y = 2;

if (greeting.length() != 4 && x == 5) {
cout << "Awesome" << endl;
}
else {
cout << "not awesome" << endl;
}

if (y % 2 == 1 || x % 2 == 1) {
cout << "One of the number is odd...Or maybe both" << endl;
}
else {
cout << "Both are even!!" << endl;
}
}
```

NOTES

- Project 1 is posted
  - Due: **9:00 PM Thursday, October 15**
- Comment your code
- Do not forget that there is a report to be submitted along with your code
  - Obstacles
  - Test Cases
- Do one thing at a time (incremental development)