Introduction to C++

CS 16: Solving Problems with Computers I
Lecture #2

Ziad Matni
Dept. of Computer Science, UCSB

A Word About Registration for CS16

FOR THOSE OF YOU NOT YET REGISTERED:

- **There's a LONG waitlist to add this class!**
    - **We now have a few openings and I will go by the prioritized waitlist**
    - **Most people on the waitlist will not get in today**

→ **WAITLISTED STUDENTS MUST SEE ME AFTER CLASS** ←

- If you are **not** on the waitlist, you will not get into this class this quarter

Lecture Outline

- Variables and Assignments

- Data Types and Expressions

- Input and Output

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

3    int main( )
4    {
5    ←→int number_of_pods, peas_per_pod, total_peas;

6    ←→cout << "Press return after entering a number.\n";
7    ←→cout << "Enter the number of pods:\n";
8    ←→cin >> number_of_pods;
9       cout << "Enter the number of peas in a pod:\n";
10      cin >> peas_per_pod;

11      total_peas = number_of_pods * peas_per_pod;

12      cout << "If you have ";
13      cout << number_of_pods;
14      cout << " pea pods\n";
15      cout << "and ";
16      cout << peas_per_pod;
17      cout << " peas in each pod, then\n";
18      cout << "you have ";
19      cout << total_peas;
20      cout << " peas in all the pods.\n";

21      return 0;
22   }
```

Note the use of tabbed spaces

```
Press return after entering a number.
Enter the number of pods:
10
Enter the number of peas in a pod:
9
If you have 10 pea pods
and 9 peas in each pod, then
you have 90 peas in all the pods.
```

1-4:	Program start
5:	Variable declaration
6-20:	Statements
21-22:	Program end

cout << "some string or another" ;
//output stream statement

cin >> some_variable;
//input stream statement

cout and cin are *objects* defined in the library *iostream*
// *means the following line is a comment*

Program Style

- Program's **layout** is designed mainly to make it **readable** by humans

- Compilers accept almost any patterns of line breaks and indentations!
  - So layout *conventions* are there not for the machine, but for the human

- Conventions have been established, for example:
  1. Place opening brace '{' and closing brace '}' on a line by themselves
  2. Indent statements (i.e. use tabbed spaces)
  3. Use only one statement per line

Some C++
Rules and Conventions

- Variables are declared ***before*** they are used
  - Typically at the beginning of program

- Statements (not always lines) ***end with a semi-colon   ;***

- Use curly-brackets { … }
                           to encapsulate groups of statements that belong together
  - Parentheses (…) have a different use in C++
  - As do square brackets […]
  - They are not interchangeable!

Some C++ Rules and Conventions

- ***Include directives*** (like `#include <iostream>`) always placed in beginning of the program before any code
  - Tells the compiler ***where to find*** information about objects used in the program

- `using namespace std;`
  - A statement that tells the compiler to use names of objects in `iostream` in a "standard" way
    - More on this in a later class

- `main` functions end with a "`return 0;`" statement

Reminder: What are Variables

- A **variable** is a *symbolic* reference to data

- The variable's **name** represents *what* information it contains

- They are called "**variables**" because the *data can change* while the **operations** on the variable remain the same

- If variables are of the same ***type***, you can perform ***operations*** on them

Variables in C++

- In C++,
  variables are placeholders for memory locations in the CPU

- We can assign a value to them

- We can change that value stored

- BUT we cannot erase the memory location of that particular variable

Types of C++ Variables: General

- There are 3 properties to a variable:
  Variables have a **name (identifier)**, a **type**, and a **value** attached to them

- **Integers**
  – Whole numbers
  – Example: 122, 53, -47

- **Floating Point**
  – Numbers with decimal points
  – Example: 122.5, 53.001, -47.201

- **Boolean**
  – Takes on one of two values:
    "true" or "false"

- **Character**
  – A single alphanumeric
  – Example: "c", "H", "%"
    - Note the use of quotation marks

- **String**
  – A string of characters
  – Example: "baby", "what the !@$?"
    - Note the use of quotation marks

There are many other types of variables

About Variable Names

- *Good* **variable name**: indicates what data is stored inside it

- *They should make sense to a non computer programmer*
  - Avoid generic names, like "var1" or "x"

- Example:

  name = "Bob Roberts"   is not descriptive enough, but
  candidate_name = "Bob Roberts"   is better

Variable Name Rules in C++

Variable names in C++ must adhere to certain rules.

- **They MUST start with either a letter or an underscore (_)**

- They cannot start with a number

- The rest of the letters can be alphanumerics or underscores.

- They cannot contain spaces or dots or other symbols

Reserved Keywords

- Used for specific purposes by C++
- Must be used as they are defined in C++
- Cannot be used as identifiers

EXAMPLE:

You cannot call a variable "**int**" or "**else**"

For a list of all C++ keywords, see:

http://en.cppreference.com/w/cpp/keyword

Declaring Variables

- Variables in C++ must be declared ____*before*____ they are used!

Declaration syntax: **Type_name** *Variable_1 , Variable_2, . . . ;*

> **NOTE:**
> One type of variable is declared at a time

Examples:

```
double  average, m_score, total_score;
int  id_num, height, weight, age, shoesize;
int  points;
```

Initializing/Assigning Variable Values

- When you declare a variable, it's not created with any value in particular

- It is good practice to initialize variables before using them
  - Otherwise they will contain whatever value is in that memory location

EXAMPLE:

```
int num, doz;
num = 5;
doz= num + 7;
```

num is initialized to 5

doz is initialized to (num + 7)

- C++ allows alternative ways to initialize variables as they are declared:

```
int num = 5, doz = 12;          or              int num(5), doz(12);
```

Assignment vs. **Algebraic** Statements

- C++ syntax is NOT the same as in Algebra

EXAMPLE:
number = number + 3
Is valid C++, but an impossible statement in algebra (0 = 3 ?!?!?!?!!!!)

- In C++, it means:
  - take the *current* value of "number",
  - add 3 to it,
  - then reassign that *new value* to the variable "number"

> *C++ shortcut:*
> ### **number += 3**

Variable Comparisons

- When variables are being **compared** to one another, we use **different symbols**
- a is equal to b   a == b
- a is not equal to b   a != b
- a is larger than b   a > b
- a is larger than or equal to b   a >= b
- a is smaller that b   a < b
- a is smaller than or equal to b   a <= b

Note:
The outcome of these comparisons are always either true or false

i.e. Boolean

Variable Types in C++
1. Integers

int: Basic integer (whole numbers, positive *OR* negative)

- Usually 32 or 64 bits wide
  - So, if it's 32 bits wide (i.e. 4 bytes), **the range is -2^{31} to $+2^{31}$**
    Which is: -2,147,483,648 to +2,147,483,647

- You can express even larger integers using:
long int and **long long int**

- You can express only positive integers using:
unsigned int

Variable Types in C++
2. Real (rational) numbers

double: Real numbers, positive *OR* negative

Type **double** can be written in two ways:
- *Simple form* must include a decimal point
  - Examples: 34.1, 23.0034, 1.0, -89.9

- Alternate form: *Floating Point Notation* (Scientific Notation)
  - **3.41e1**   means 34.1
  - **3.67e17**   means 367000000000000000.0   (17 digits after "3")
  - **5.89e-6**   means 0.00000589   (6 decimal places before "5")

- Number **left of e** (for exponent) <u>does not</u> require a decimal point
- The exponent <u>cannot</u> contain a decimal point

Variations on Number Types

- long int

- long double

- short int

- float   (a shorter version of "double")

Variable Types in C++
3. Characters

char: single character

- Can be any single character from the keyboard
- To declare a variable of type char:

<div align="center">

`char letter;`

</div>

- Character constants are enclosed in <u>single</u> quotes

<div align="center">

`char letter = 'a';`

</div>

Variable Types in C++
4. Strings

string: a collection of characters (a *string* of characters)

- **string** is a *class*, different from the primitive data types discussed so far.
  - We'll discuss <u>classes</u> further in the course

- Using strings requires you to include the "string" module:

  #include <string>

- To declare a variable of type string:

  string name = "Homer Simpson";

Note on ‘ vs “

- Single quotes are only used for **char** types
- Double quotes are only used for **string** types

- So, which of these is ok and which isn't?
  ```
  char letter1 = "a";
  char letter2 = 'b';
  string town1 = "Mayberry";
  string town2 = 'Xanadu';
  ```

Type Compatibilities

- General Rule: **You cannot operate on differently typed variables.**

- In general, store values in variables of the *same* type, so that you can operate on them later.

- The following is an example of a type mismatch:
    ```
    int my_var;
    my_var = 2.99;
    ```

- *If* your compiler allows this,
        my_var will most likely contain the value 2, not 2.99

int ←→ double

- Variables of type *double* should **not** be assigned to variables of type *int*

- Variable of type *int*, however, **can** normally be stored in variables of type *double*

EXAMPLE:

```
double numero;
numero = 2;
```

- *numero* will contain 2.0

Variable Types in C++
5. Booleans

bool: a binary value of either "true" (1) or "false" (0).

- You can perform LOGICAL operations on this type:
  - || Logical OR
  - &&   Logical AND
  - | Bitwise OR
  - &   Bitwise AND
  - ^   Bitwise XOR

- Also, when doing comparisons, the result is a Boolean type.

EXAMPLE: What will this print out??

```
int a = 44, b = 9;
bool c;
c = (a == b);
cout << c;
```

Ans: 0

Arithmetic Operations on Numbers

- Arithmetic operators can be used with any numeric type
  - Usual types of operations: **+   -   *   %   (for int)**
  - Usual types of operations: **+   -   *   /   (for double)**
  - Use brackets (...) to ensure required flow of operation

- An *operand* is a number or variable used by the operator

- Result of an operator *depends on the types of operands*
  - If both operands are **int**, the result is **int**
  - If one *or* both operands are **double**, the result is **double**

Division of Type **double**

- Division with at least 1 operator of type double produces the expected results.

```
double divisor, dividend, quotient;
divisor = 3;
dividend = 5;
quotient = dividend / divisor;
```

   quotient will be **1.6666…**

- Result is the same if *either* dividend or divisor is of type int

Division of Type **int**

- Don't do this operation (for serious purposes)
- Division between two int types, results in an int answer

```
int divisor, dividend, quotient;
divisor = 3;
dividend = 5;
quotient = dividend / divisor;
```

  quotient will be **1**, *not 1.6666…*

- Integer division **does not round the result**!
  Instead, the fractional part is discarded!

Modulo Operator (%)

- Shows you the remainder of a division between two **int** types

```
int divisor, dividend, remainder;
divisor = 4;
dividend = 15;
remainder = dividend % divisor;
```

What value will variable "remainder" have?

Ans: 3

Arithmetic Expressions

- Precedence rules for operators are the same as what you used in your algebra classes
  - EXAMPLE: x + y * z   (y is multiplied by z first)

- Use parentheses to force the order of operations (recommended)
  - EXAMPLE: (x + y) * z   (x and y are added first)

Operations on Variables

- Certain operations only work with certain variable types
  - <u>NOTE:</u> C++ compilers are not always consistent with this…

Examples:

- Say you have 6 vars:
  `A = 1, B = 2.0, C = "head ", D = "and shoulders",  E = true,  F =`
  `false`
  Integer   Double   Strings   Booleans

- Can you do the following in C++?:   ✓ *Yes*
  - A + B   ✓ *Yes*   – B - A   ✓ *No*
  - A * B   ✓ *Yes*   – C - D   ✓ *No*
  - C + D   ✓ *Yes, BUT NOT RECOMMENDED!!!*   – C + E   ✓ *Yes*
  - A + E   ✓ *Yes*   – (E || F) && E
  - E && F

Matni, CS16, Sp17

Operator Shorthands

- Some expressions occur so often that C++ contains shorthand operators for them

- All arithmetic operators can be used this way:

  - **count = count + 2;**   ---can be written as--- **count += 2;**
  - **bonus = bonus * 2;**   ---can be written as--- **bonus *= 2;**
  - **time = time / factor;**   ---can be written as--- **time /= factor;**
  - **remainder = remainder % (cnt1+ cnt2);**
                                     ---can be written as--- **remainder %= (cnt1 + cnt2);**

Review of Boolean Expressions:
AND, OR, NOT

AND operator   **&&**

- (expression 1) && (expression 2)
- True if **_both_** expressions are true

OR operator   **||**

- (expression 1) || (expression 2)
- True if **_either_** expression is true

Note: no space between each '|' character!

NOT operator   **!**

- !(expression)
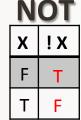- False, if the expression is True (and vice versa)

Truth Tables for Boolean Operations

AND

X	Y	X && Y
F	F	F
F	T	F
T	F	F
T	T	T

OR

X	Y	X || Y
F	F	F
F	T	T
T	F	T
T	T	T

NOT

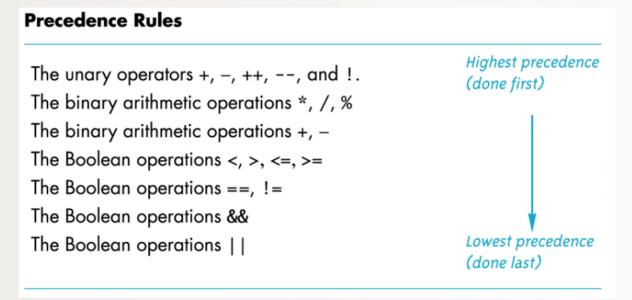| X	! X
F	T
T	F

IMPORTANT NOTES:

1. AND and OR are **not opposites** of each other!!
2. AND: if *just one* condition is false, then the outcome is false
3. OR: if *at least one* condition is true, then the outcome is true
4. AND and OR are **commutative, but not when mixed** (so, order matters)

    X && Y = Y && X

    X && (Y || Z)   is not the same as   (X && Y) || Z

Precedence Rules on Operations in C++

- If parenthesis are omitted from C++ expressions, the default precedence of operations is:

Precedence Rules

The unary operators +, −, ++, −−, and !.
The binary arithmetic operations *, /, %
The binary arithmetic operations +, −
The Boolean operations <, >, <=, >=
The Boolean operations ==, !=
The Boolean operations &&
The Boolean operations ||

Highest precedence (done first)

Lowest precedence (done last)

Inputs and Outputs

Data Streams - Definitions

- **Data stream:** a sequence of data
  - Typically in the form of characters or numbers

- **Input stream:** data for the program to use
  - Typically originates at the keyboard, or from a file

- **Output stream:** the program's output
  - Destination is typically the monitor, or to a file

cout and cin

- Output and input stream **objects** ; very popularly used in C++
- To make the definitions of **cin** and **cout** available to a program, you have to declare the statement:

```
#include <iostream>
```

- Using *directives* like that usually includes
  a collection of *defined names*.

- To make the objects **cin** and **cout** available to our program, you should also declare the statement:

```
using namespace std;
```

Examples of Use (cout)

```
cout << number_of_bars << " candy bars\n";
```

- This sends two items to the monitor (display):
  - The value of **number_of_bars**
  - The quoted string of characters **" candy bars\n"** (note the starting space)
  - The '**\n**' causes a **new line** to be started following the 's' in bars

- A new *insertion operator* **(<<)** must be used **for each item of output**

- <u>Note</u>: do **<u>not</u>** use single quotes for the strings (more on that later)

Escape Sequences

- Tell the compiler to treat certain characters in a special way
  - **** (back-slash) is the escape character

- Example: To create a newline in the output, we use
  - **\n** – as in, **cout << "\n";**
  - An alternative: **cout << endl;**

- Other escape sequences:
  - **\t**   horizontal tab character
  - ****   backslash character
  - **\"**   quote character
  - **\a**   audible bell character

For a more complete list of escape sequences in C++, see:

http://en.cppreference.com/w/cpp/language/escape

Formatting Decimal Places

A common requirement when displaying numbers.

EXAMPLE: Consider the following statements:

```
double price = 78.5;
cout << "The price is $" << price << endl;
```

- Do you want to print it out as:

```
The price is $78.5
The price is $78.50
The price is $7.850000e01
```

> Likely, you want the 2nd option
> You have to DEFINE that ahead of time

Formatting Decimal Places with **cout**

- To specify fixed point notation, use:

  <div align="center">

  `cout.setf(ios::fixed)`

  </div>

- To specify that the decimal point will always be shown

  <div align="center">

  `cout.setf(ios::showpoint)`

  </div>

- To specify that *n* decimal places will always be shown

  `cout.precision(n)`   --- *where n can be 1, 2, 3, etc…*

EXAMPLE:
```
double price = 78.5;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout << "The price is " << price << endl;
```

> You usually only need to do this ONCE in a program, unless you decide to change the format

Inputs via **cin**

- **cin** is an input stream bringing data from the keyboard
- The *extraction operator* (**>>**) removes data to be used and can be used more than once

EXAMPLE:
```
cout << "Enter the number of bars in a package\n";
cout << " and the weight in ounces of one bar.\n";
cin >> number_of_bars;
cin >> one_weight;
```
Alternative: `cin >> number_of_bars >> one_weight;`

- This code prompts the user to enter data then reads 2 data items from **cin**
- The 1st value read is stored in *number_of_bars*, the 2nd value in *one_weight*
- Data entry can be separated by spaces OR by return key when entered

Entering Multiple Data Input Items

- Multiple data items are ***best*** separated by spaces
- Data is not read until the **Enter** key is pressed
    - This allows user to make corrections

EXAMPLE:
```
cin >> v1 >> v2 >> v3;
```
*Requires **3** space separated values **or** an enter in between each value*

- So, user might type:
```
34 45 12<enter key>   or   34<enter key>45<enter key>12<enter key>
```

Space chars.

Design Recommendations with I/O

- First, prompt the user for input that is desired
  - Use **cout** statements provide instructions

  ```
  cout << "Enter your age: ";
  cin >> age;
  ```

- Note: absence of a new line before using cin
  - Why?

- Then, echo the input by displaying what was read
  - This gives the user a chance to verify the data entered

  ```
  cout << age << " was entered." << endl;
  ```

YOUR TO-DOs

❑ Finish Lab1 by Friday

❑ Do HW1 and hand it in on Thursday in class

❑ Visit Prof's and TAs' office hours if you need help!

❑ Reverse global warming
  ❑ Bonus points for ending world hunger

</LECTURE>