C++ Basics 1

CS 16: Solving Problems with Computers I
Lecture #3

Ziad Matni
Dept. of Computer Science, UCSB

Announcements

- **Homework #2 due today**
  - Please take out any staples or paper clips
- **Lab #1 is due on Friday AT NOON!**
  - Use submit.cs

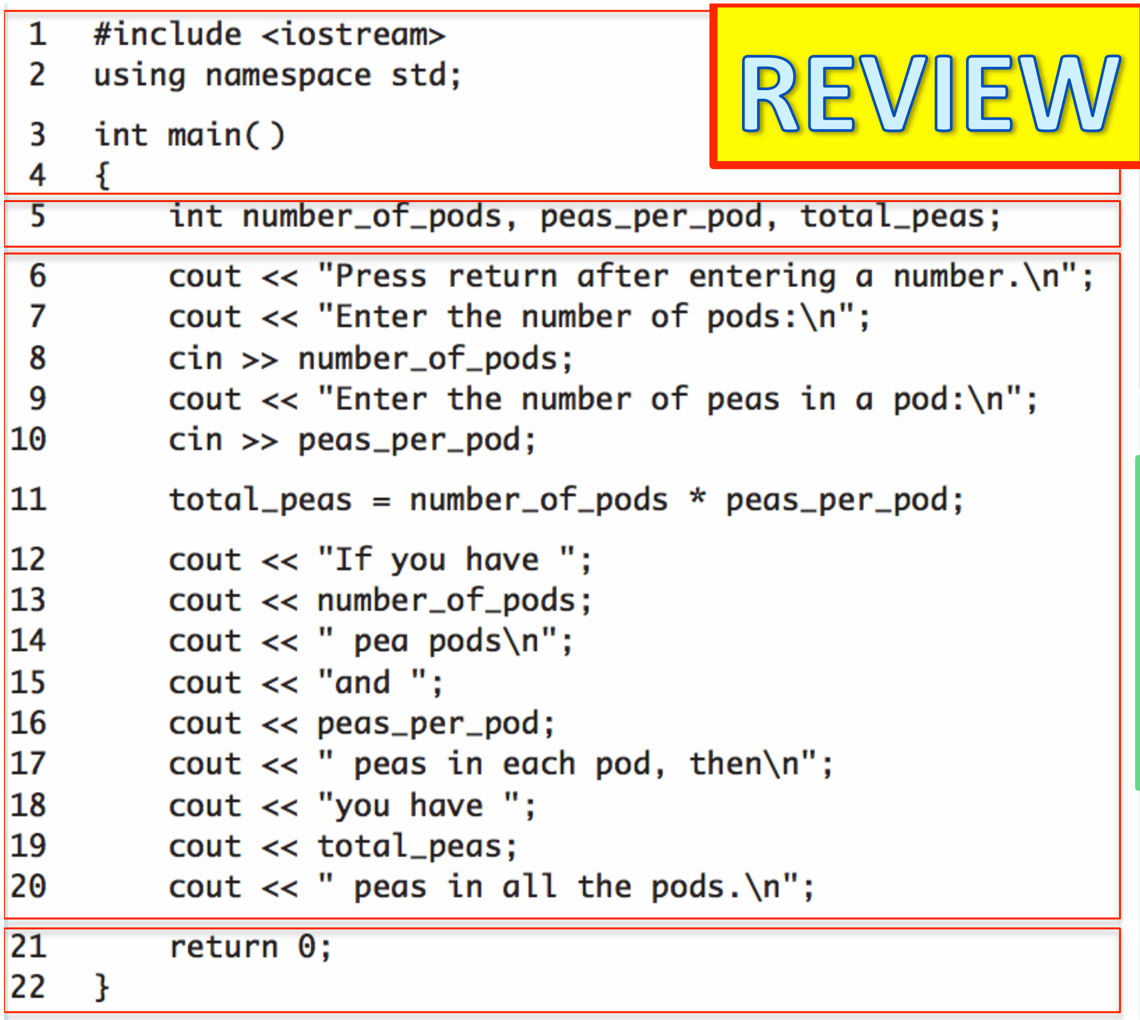
- **Class is closed to new registration**

- **No more switching lab times**
  - Labs at 9am, 10am, 11am, 12pm are **FULL**
  - Other labs have some space left

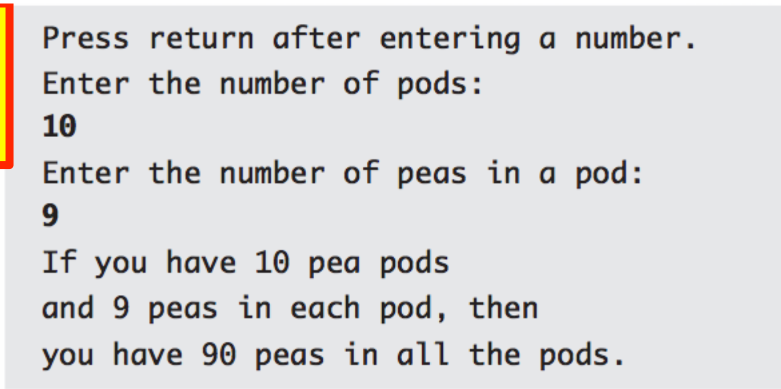
Lecture Outline

- Variables and Assignments
- Input and Output
- Data Types and Expressions

REVIEW



```
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  }
```



```
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*

stream is an entity where a program can either insert or extract characters

cout* and *cin* are objects defined in *iostream

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
  - The variables “a” and “b” can take on different values, but I may always want to add them together

Variables

96
SCORE
“A”
GRADE

- Variables are like “buckets” that can keep data
  - You can label these buckets with a **name**
  - When you reference a bucket, you use its name, not the data stored in the bucket
  - You can “re-use” the buckets

- If two 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 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”

- There are many other types of variables

- 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

Types of Variables: General

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

Generally, in most HL computer languages, there are the following types of variables:

- **NUMBERS**
  - Either integers or real numbers (called "double" in C++)
- **LETTERS**
  - Characters or *strings* of characters
- **LOGICAL**
  - Takes on one of two values: "true" or "false" (called Boolean)

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
- Example:

  name = “Bob Roberts” is not descriptive enough,

  but

  candidate_name = “Bob Roberts” is better

About Variable Names

- The **name of the variable is not the information!**
  - Example: class_size = 40 is good, but
    class_size_is_40 = TRUE is bad

- Variable names must adhere to certain rules.
  In C++,
  they MUST start with either a letter or an underscore (_)

- The rest of the letters can be alphanumerics or underscores.
- They cannot start with a number
- They cannot contain spaces or dots or other symbols

Keywords

- Also called **reserved words**
- 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 must be declared ___*before*___ they are used in a program!

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 Variables

- When you declare a variable,
  it's not created with any value in particular
- You HAVE to initialize variable before using them

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 Statements

- How one changes the value of a variable.

```
total_weight = 12 * one_weight ;
```

Note:

- Assignment statements always end with a semi-colon

Assignment Statements vs. Algebraic Statements

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

EXAMPLE:

number = number + 3

- Is 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”

Assignments vs. Comparisons

Variables can be assigned as:

- Declarations e.g. *a = 6*, or *name = “Buddy”*
- Calculations e.g. *c = a + b*

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

<u>Note:</u>
The outcome of these comparisons are always either true or false

Inputs and Outputs

Data Streams

- 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 have to 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

- Note: a new insertion operator is used **for each item of output**

- Note: do not 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 (new to later versions of C++):
  - `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're going to have to DEFINE that ahead of time

Formatting Decimal Places with **cout**

- To specify fixed point notation, use:

```
cout.setf(ios::fixed)
```

- To specify that the decimal point will always be shown

```
cout.setf(ios::showpoint)
```

- 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;
```

Inputs via **cin**

- cin is an input stream bringing data from the keyboard
- The extraction operator (>>) removes data to be used

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;
```

- This code prompts the user to enter data then reads 2 data items from **cin**

- The first value read is stored in *number_of_bars*
- The second value read is stored 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
- So, user might type:

```
34 45 12 <enter key>
```

Design Recommendations

- 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;
```

Data Types

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`: 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:

```
char letter;
```

- Character constants are enclosed in single quotes

```
char letter = 'a';
```

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 classes further in the course

- So, using strings requires the following be added to the top of your program:

```
#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 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*
  - Example from Homework #2

- 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

More on these later…

- 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;
```

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 one 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 (see Homework #2).

```
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**, rather the fractional part is discarded!

Modulo Operator (%)

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

```
int divisor, dividend, remainder;
divisor = 3;
dividend = 5;
remainder = dividend % divisor;
```

What value will “remainder” will be?

Arithmetic Expressions

- Precedence rules for operators are the same as what you used in your algebra classes
  - Anyone remember junior high???
  - 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

- You *should* only perform operations on **<u>same-type variables</u>**
- Certain operations only work with certain variable types

<u>Examples:</u>

- Say you have 6 variables:

  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++?:

– A + B	– **Yes**
– A * B	– **Yes**
– C + D	– **Yes**
– A + E	– **Yes, BUT NOT RECOMMENDED!!!**
– E && F	– **Yes**

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);**

TO DOs

- Readings
  - The rest of Chapter 2, of textbook
- Homework #3
  - Due on Tuesday, 10/4
  - Submit in class
- Lab #2
  - Prepare for it by reading the handout
  - Made available to you by Friday evening

</LECTURE>