

# **Introduction to C++**

## **General Rules, Conventions and Styles**

**CS 16: Solving Problems with Computers I**  
**Lecture #2**

Ziad Matni  
Dept. of Computer Science, UCSB

# Administrative

- This class is currently **FULL** and the waitlist is **CLOSED**
  - Will not be adding anyone else *Please do not ask again*
- Lab #1 and *submit.cs* issues
- Homework #1 and working on GauchoSpace
- Reminder: Don't leave your valuables behind in the lab (or class)!

# Lecture Outline

---

- Basic Rules and Conventions of C++
- 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      Note the use of tabbed spaces  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*

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

# What's The Difference???

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

int main( )
{
    int n = 5;
    while (n < 10)
    {
        cout << n;
        n = n + 1;
    }

    return 0;
}
```

```
#include <iostream>
using namespace std;int main( )
{int n=5;while (n<10)
{cout<<n;n=n+1;}return 0;}
```

*A compiler program can read either one!*

*But which one can YOU read better?!?! 😊*

# Program Style

*We will check for this convention use in your lab assignments!*

- The **layout** of a program is designed mainly to make it **readable** by humans
- C++ Compilers accept almost any patterns of line breaks and indentations!
  - So layout *conventions* are there not for the machine, but for the human
  - Convention vs. Rules – what's the difference??
- Conventions have been established, for example:
  1. Place opening brace '{' and closing brace '}' on a line by themselves
  2. Use indented statements (i.e. use tabbed spaces)
  3. Use only one statement per line



# Some C++ Rules and Conventions

*Breaking these rules  
is considered a  
syntax error;  
your program  
won't compile!*

- 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>`) are always placed in the beginning of the program before any code
  - Tells the compiler **where to find** information about objects used in the program
- `using namespace std;`
  - Tells the compiler to use names of objects in `iostream` in a “standard” way
- `main` functions end with a “`return 0;`” statement
  - You should always have this – although it’s a convention, not a strict rule



## 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 – you also make your own types!

# About Variable Names

*We will check for this convention use in your lab assignments!*

- **Good variable name:** indicates what data is stored inside it
  - A good variable name is a “noun” or “noun phrase”, e.g.: FirstName
  - A good function name is a “verb” or “verb phrase”, e.g.: SortNumbers()
- *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++

*Breaking these rules  
is considered a  
syntax error:  
your program  
won't compile!*

*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 alphanumeric or underscores.
- They cannot contain spaces or dots or other symbols
- Which of these is a legal variable name in C++

**4MyBae**

**\_StopCondition**

**MyLittlePony\_007**

**James.Bond**



# Variable Name Casing

*We will check for this convention use in your lab assignments!*

*When naming variables, functions, etc...*

- **Snake Case:** Using underscore character ('\_')
  - Example: `mortgage_amount` `function_fun()`
  - Associated with C, C++ programmers
- **Camel Case:** Using upper-case letters to separate words
  - Example: `MortgageAmount` `FunctionFun()`
  - Associated with Java programmers
- For this class, YOU CAN USE EITHER! **But PICK ONE AND BE CONSISTENT!!!**



# Reserved Keywords

*Breaking these rules  
is considered a  
syntax error:  
your program  
won't compile!*

- 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>

## Other Styling Conventions

*We will check for this convention use in your lab assignments!*

- Comments: Must have them
  - In C++, use `//` for one line at a time, or `/* ... */` for multiple lines
- Tabbing and Braces:
  - Code inside of `main()` must be tabbed appropriately
    - Even one-liner if-statements
  - Open and close curly braces `{...}` on new lines and align them with the block

## Example of Good Styling

```
int main()
{
    // Get user input on number of people
    // Then determine if there is room for them
    int max_capacity(100), num_people;
    cout << "Enter number of people: ";
    cin >> num_people;
    if (num_people > max_capacity)
    {
        cout << "Too many people! By a count of ";
        cout << num_people - max_capacity;
    }
    else
    {
        cout << "Ok!";
    }
    return 0;
}
```

# Declaring Variables

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

Declaration syntax:     **Type\_name** *Variable\_1* , *Variable\_2* , ... ;

## Examples:

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

# Initializing/Assigning Variable Values

*Using = or ( )  
for assignment of  
declared values  
is up to you!*

- 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**
- Do not declare variables inside loops!!!

## EXAMPLE:

```
int num, doz;  
num = 5;  
sum(5);  
doz = num + 7;
```

**num** is initialized to 5

and so is **sum**

**doz** is initialized to (num + 7)

## Assignment vs. Algebraic Statements

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

EXAMPLE:

**number = number + 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**

*Also works with:*  
**-= \*= /= %= etc...**



# 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 than 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*

**Boolean variables:**

**false** = 0

**true** ≠ 0

**(note the lower-case)**

# 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, **the range is  $-2^{31}$  to  $+2^{31} - 1$**   
Which is: -2,147,483,648 to +2,147,483,647
- You can express even larger (+ve and -ve) integers using:  
**long int** and **long long int**
- You can express only positive integers (and thus get a longer +ve range) 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 3670000000000000000.0 (17 digits after “3”)
  - **5.89e-6** means 0.00000589 (6 decimal places before “5”)
- Number **left of e** (for exponent) does not require a decimal point
- The exponent cannot contain a decimal point

## 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
- Using C++ strings requires you to include the “string” module:  
`#include <string>`
- To declare a variable of type string:  
`string name = “Homer Simpson”;`
- There are “older” types of strings called **C-Strings** that are still in use in C++
  - More on those later...



## 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.
  - Except with int and double types
  - Just like in most computer languages
- So, if:  
`int my_var = 2;`  
`char my_char = 'x';`  
then:  
`my_var + my_char` *is a syntax error*
- There are rules with operations between **int** and **double**...

# int $\leftrightarrow$ 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 2.0000 (unfixed number of places after decimal pt)

EXAMPLE:     int numero;  
                  numero = 2.789;

- *numero* will contain 2

If one or both operands are **double**, the result is **double**

int  $\leftrightarrow$  double

So, what happens with variable **z** here?

```
int x(9);  
double y(4), z;  
z = x / y;  
cout << z;
```

**This prints out: 2.25**

So, what happens with variable **p** here?

```
int n(4);  
double m(9), p;  
p = m / n;  
cout << p;
```

**This prints out: 2.25**

So, what happens with variable **c** here?

```
int a(9), b(4);  
double c;  
c = a / b;  
cout << c;
```

**This prints out: 2**

# 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

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 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)

# 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)*

# YOUR TO-DOs

---

- ☐ Finish Lab1 by Monday
- ☐ Do HW2 by Tuesday
  
- ☐ Visit Prof's and TAs' office hours if you need help!
  - Prof.'s hours are **MONDAY from 11 AM to 12 PM** (or by appointment!)
  
- ☐ Reverse global warming
  - ☐ Bonus points for ending world hunger

**</LECTURE>**