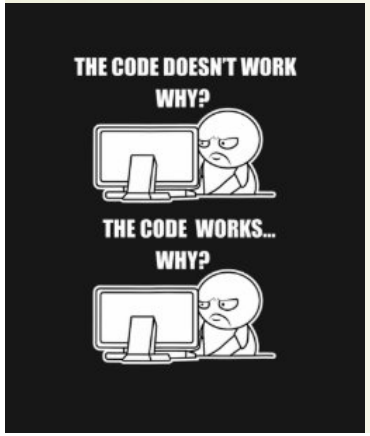
The Basics of C++

CS 16: Solving Problems with Computers I
Lecture #2

Ziad Matni
Dept. of Computer Science, UCSB




```
int main(int argc, char *argv[])
{
    if (argc > 1)
        filename = argv[1];
    ifstream setIn(filename);
    ifstream vecIn(filename);
    set<string> wordSet = getWordSet(setIn);
    vector<string> wordVec = getWordVec(vecIn);
    map<string, string> wordMap = generateMap(wordVec);

    string name = filename.substr(0, filename.size() - 4);
    string setFilename = name + "_set.txt";
    string vecFilename = name + "_vec.txt";
    string mapFilename = name + "_1_1.txt";

    // Writes set file
    ofstream setOut(setFilename);
    for (set<string>::iterator it = wordSet.begin(); it != wordSet.end(); it++)
    {
        setOut << *it << endl;
    }
    setOut.close();

    // Writes vector file
    ofstream vecOut(vecFilename);
    for (int i = 0; i < wordVec.size(); ++i)
    {
        vecOut << wordVec[i] << endl;
    }
    vecOut.close();

    // Writes to map
    ofstream mapOut(mapFilename);
    printMap(wordMap, mapOut);
    mapOut.close();

    // Generate and print random string
    string str = "";
    for (int i = 0; i < 100; i++)
    {
        cout << wordMap[str] << " ";
        str = wordMap[str];
    }
    cout << endl << endl << endl;

    // Generate more intelligent map
    map<string, vector<string>> wordVecMap;
    str = "";
    for (int i = 0; i < wordVec.size(); i++)
    {
        wordVecMap[str].push_back(wordVec[i]);
        str = wordVec[i];
    }
```

Administrative

Important changes from last week

- Re: Attendance & Participation
- Re: Grading Scheme
- Re: ULAs
- Re: Prof's Office Hours

Administrative

- Homework 1 will be posted after lecture
  - Rules for submission on Gradescope
- Lab 1 was posted yesterday (Monday)
  - Remember, you have lab today (1, 2, 3, 4 pm)
  - Re: the use of IDEs like MS VSCode, or others…
- Both assignments are due next Monday
- If unsure on how to submit:
  (a) Read the instructions
  (b) Post/Read on Piazza
  (c) Ask us during office hours and ULA Peer Mentoring Sessions

High-Level Computer Languages

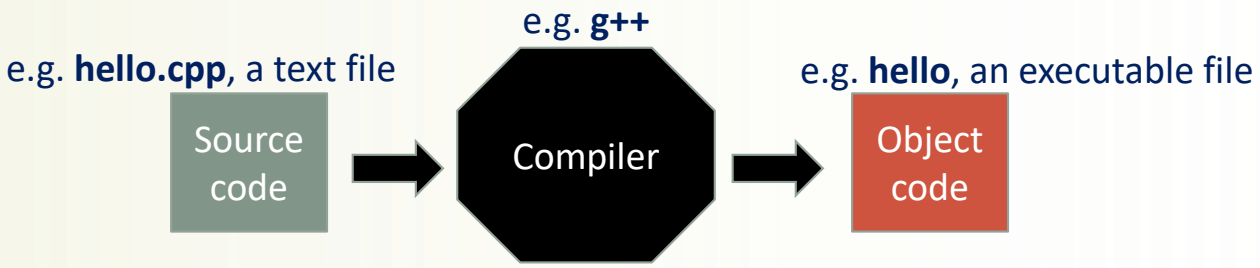
- A computer language that closely mimics "natural language"
  - As opposed to just being 0s and 1s (that's called "machine language")
- ***High-level languages*** provide *high abstraction* to the CPU Instructions
  - Your programs very much look like ***algorithms***
- A program that "translates" a High Level Language into ***Low Level Language*** (like machine language) is called a ***compiler***
  - Why are compilers necessary???
  - ***Because CPUs ONLY understand their instructions in Machine Language***

Compilers

- Language-specific
  - Compiler for Python will not work for C++, etc…
- Linux/UNIX OS have different built-in compilers
  - e.g. ***g++*** for C++, ***clang*** for C, etc…

- Source code
  - The original program in a high level language (text file)

- Object code
  - The translated version in machine language (binary file)



Introduction to the C++ Language

When was it invented?

In the 1980s…

Was it based on anything else?

The C Language

Is it still popular?

Yes! In the top-3 of most used today with Python and Java

A Sample C++ Program

A common and simple C++ program begins this way:

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

int main() {
// This is a comment (optional)
```

And ends this way

```
 return 0;      // Return a value 0 to the Oper.Sys. (code for “all ok”)
}
```

Standard Input / Output

- When sending variable data to standard output (i.e. screen):

```
std::cout << data;
```

BUT! If you have the **using namespace std;** *instruction outside of* **main()**, *then just do this:*

```
cout << data;
```

- When getting data from standard input (i.e. keyboard)

(and assigning it to a variable called "data"):

```
cin >> data;
```

(or **std::cin** as in the example above)

```
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  }
```

Declaring integer variables

Note the semicolons!!

```
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*

Mind the Syntax!!

```
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

```
g a number.


n 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. **Use indented statements (i.e. use tabbed spaces)**
  2. **Use only one statement per line**
  3. **Use opening/closing curly braces { … } cleanly**

Some C++ **Rules** and Conventions

*Breaking these rules is considered to be 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 main 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`** (like `cout` or `cin`) in a "standard" way
  - Otherwise, you'd have to explicitly precede these objects by their namespace
    - Example, you'd have to use **`std::cout`** instead of just **`cout`**

Some C++ Rules and **Conventions**

- `main` functions end with a "`return 0;`" statement
  - You should always have this – although it's a convention, not a strict rule

- General template should look like this:

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

int main()
// This is a comment (optional)
{
   // code goes here

   return 0;
}
```

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) which is tied to a memory address in the CPU**,
  a **variable type**, and a **value** attached to them

- **Integers (int)**
  - Whole numbers
  - Example: 122, 53, -47

- **Floating Point (float, double)**
  - Numbers with decimal points
  - Example: 122.5, 53.001, -47.201

- **Boolean (bool)**
  - Takes on one of two values:
    "true" or "false"

- **Character (char)**
  - A single alphanumeric
  - Example: 'c', 'H', '%'
    - Note the use of single-quotation marks

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

There are many other types of variables – you can also create your own types!

Range of Integers in C++

- The type **`int`** takes up 4 bytes of data in computer memory (by design)
- This puts a hard limit on how large you can make **`int`** types
- Range is: -2,147,483,648 to +2,147,483,647

- Variations on the **`int`** type in most computer settings
  - `unsigned int` 0 to 4,294,967,295
  - `short int` -32,768 to 32,767 ← takes up 2 bytes
  - `unsigned short int` 0 to 65,535
  - `long int` in most computers, it's the same range as `int`

- In this class, we will only (mostly) use "simple" `int`, not the other variations

Floating Point Numbers

- You can use either **`float`** or **`double`** types for F.P.
- `float`
  - called “single-precision”, allows a certain range and a certain precision
- `double`
  - called “double-precision”, allows a wider range and more precision than `float`
- The book likes to use **`double`**, and it’s fine if we use that too…

Conventions 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, even to a non-computer programmer!*
  - Avoid generic names, like "**var1**" or "**x**"
- Example:

```
int t = 60;   // time in minutes
int timeInMinutes = 60;
```

This is meh…

MUCH better name…

Rules About Variable Names in C++

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

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

- **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

- Which of these is a legal variable name in C++?

4MyBae	_StopCondition	Thing1	Pokemon_007	James.Bond42
X	✓	✓	✓	X

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

- 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

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

Other Styling Conventions

We will check for this convention use in your lab assignments!
Starting with Lab2

- 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 Bad Styling

```
int main(){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;}
```

This will still compile and run, but it is against usual styling-convention and harder to read

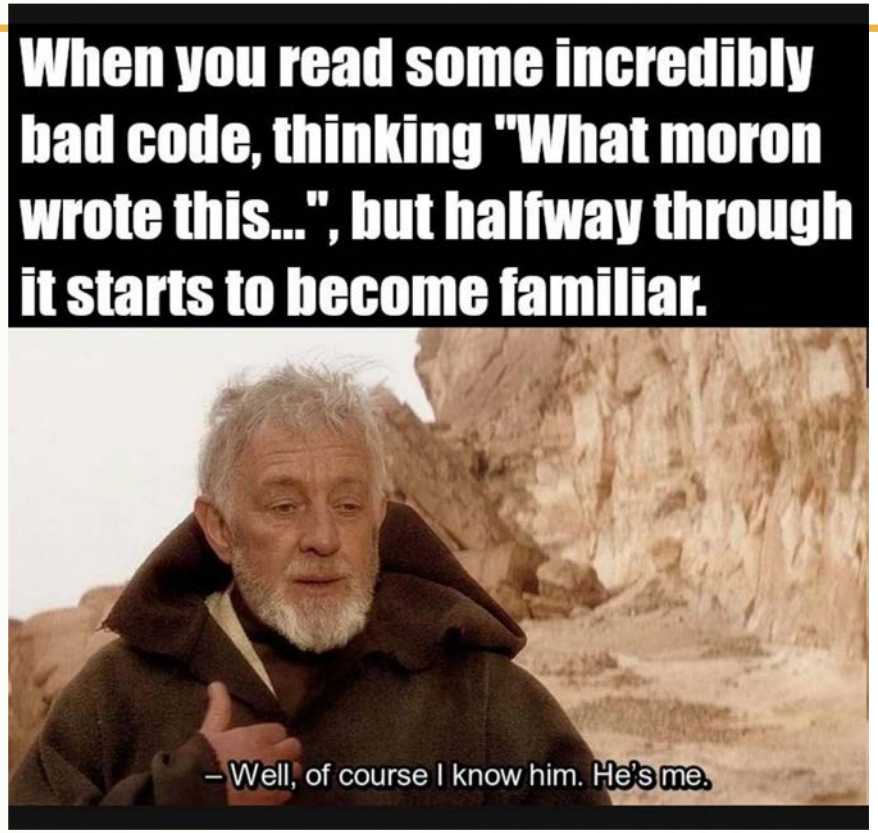
Example of Good Styling

```
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;
}
```

*Note the use of **comments**, **tabbed spaces**, having instructions on **seperate lines**, **clean** use of the {...}*
Also, it’s easier to read!

Beautiful Code Will Make YOU Happy

- In the real world, code gets reviewed by others
- You write code and then other people look at it to make sure it works
- You might even look back at your old code and be mad at past you for not writing cleaner code



Source: A.Sanchez, M.Lim, 2019

So What's the "Right" Way to Style C++ Code???

- We will pick a common convention for you to use in this class
  - Different places have slight variations on style
- Look for a post in the next few days that will detail this
- Starting with **Lab2**, we will be checking on your code styling

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;
int 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!
  - Applies to almost all programming languages…

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 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

Boolean variables:
false = 0
true ≠ 0
(note lower-case!!!)

YOUR TO-DOs

- ❑ Watch the videos that I will post on **Thursday**
- ❑ Do **Lab1** (due next week **Mon.**)
- ❑ Do **Homework1** (due next week **Mon.**)
- ❑ Do **Quiz1** this week (on **Fri.**)
- ❑ Do your readings from textbook!

- ❑ Remember Office Hours for Prof., TAs, ULAs!

- ❑ Make the world a better place
  - ❑ Start by being the change you want to see in the world! ☺

</LECTURE>