A Crash Course in Programming with C++ and the Ubuntu O.S.

Dr. Daniel A. Ray
MCS Dept
UVa-Wise

June 8, 2010
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together
C++ is a Programming Language

So... what’s a programming language

Computer’s can’t understand plain English (or any other human language, for that matter).
We have to learn how to speak so computers can understand.
But, unfortunately, computers only speak binary:
**C++ is a Programming Language**

So... what’s a programming language

Computer’s can’t understand plain English (or any other human language, for that matter).

We have to learn how to speak so computers can understand. But, unfortunately, computers only speak binary:

**Binary ASCII Code**

010010000110010101011011000110110001101101011100101110001
0000010010010010010000011000010110110100100000011000
01001000001100011011011110110110101110000011101010
111010001100101010111001000101110
C++ is a Programming Language

So... what’s a programming language

Computer’s can’t understand plain English (or any other human language, for that matter).
We have to learn how to speak so computers can understand.
But, unfortunately, computers only speak binary:

**Binary ASCII Code**

<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>01001000011001010101101100011011000110110001101111</td>
<td>&quot;Hello, I am a computer.&quot;</td>
</tr>
</tbody>
</table>

0100100001100101010110110001101111100101100001
0000001001001001000000011000010110110100100000011000
010010000001100011011011110110110101110000011101010
111010001100101010111001000101110
Do I have to learn to speak binary?

Thankfully, no.
What is C++?

Do I have to learn to speak binary?

Thankfully, no. Computer scientists have written human-understandable languages that can be automatically translated into binary code by a special program called a compiler. One such language is called C++.

Quick Amazon Search

Do I have to learn to speak binary?

Thankfully, no. Computer scientists have written human-understandable languages that can be automatically translated into binary code by a special program called a compiler. One such language is called C++.

Quick Amazon Search


In order to program our robots (which are really just small computers attached to gears, motors and wheels) we need to be able to write instructions (tell the computer what to do) using the C++ language.
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together
How do we write code?

1. Write the code and save it to a file:

2. Open a special program called a terminal and navigate to the directory where you saved your file:

3. Compile the program:
How do we write code?

1. Write the code and save it to a file:
   - Open up text editor (Applications > Programming > Geany)
   - Click the “New” button at the top to get a blank page

2. Open a special program called a terminal and navigate to the directory where you saved your file:

3. Compile the program:
Our First C++ Program

How do we write code?

1. Write the code and save it to a file:
   1. Open up text editor (Applications > Programming > Geany)
   2. Click the “New” button at the top to get a blank page
   3. Type C++ code in to the opened file: [Jump to First Program]

2. Open a special program called a terminal and navigate to the directory where you saved your file:

3. Compile the program:
How do we write code?

1. Write the code and save it to a file:
   - Open up text editor (Applications > Programming > Geany)
   - Click the “New” button at the top to get a blank page
   - Type C++ code into the opened file: [Jump to First Program]
   - Create a new folder on the Desktop (right-click, just like in Windows)
   - Save the file into your newly created folder as `{yourfilename}.cc`

2. Open a special program called a terminal and navigate to the directory where you saved your file:
   - Go to the Applications menu (top of the screen)
   - Click on the System Tools menu item
   - Click on Terminal
   - Type `pwd` in to the terminal window
   - Now type `ls` in to the terminal window
   - Inside the terminal type `cd ~\Desktop\{nameofyourfolder}\`
   - Type `pwd`, now type `ls`

3. Compile the program:
   - In the terminal type: `g++ {yourfilename}.cc`
   - Assuming our code had no errors, this will create a file called a.out
   - In the terminal type: `a.out` ... this runs the program
How do we write code?

1. Write the code and save it to a file:
   - Open up text editor (Applications > Programming > Geany)
   - Click the “New” button at the top to get a blank page
   - Type C++ code in to the opened file: [Jump to First Program]
   - Create a new folder on the Desktop (right-click, just like in Windows)
   - Save the file into your newly created folder as \{yourfilename}.cc

2. Open a special program called a terminal and navigate to the directory where you saved your file:
   - Go to the Applications menu (top of the screen)
   - Click on the System Tools menu item
   - Click on Terminal
   - type pwd in to the terminal window ... now type ls in to the terminal window

3. Compile the program:
How do we write code?

1. Write the code and save it to a file:
   1. Open up text editor (Applications > Programming > Geany)
   2. Click the “New” button at the top to get a blank page
   3. Type C++ code in to the opened file: ▶ Jump to First Program
   4. Create a new folder on the Desktop (right-click, just like in Windows)
   5. Save the file into your newly created folder as {yourfilename}.cc

2. Open a special program called a terminal and navigate to the directory where you saved your file:
   1. Go to the Applications menu (top of the screen)
   2. Click on the System Tools menu item
   3. Click on Terminal
   4. type pwd in to the terminal window ... now type ls in to the terminal window
   5. Inside the terminal type cd ~ \Desktop\{nameofyourfolder}
   6. type pwd, now type ls

3. Compile the program:
Our First C++ Program

How do we write code?

1. Write the code and save it to a file:
   1. Open up text editor (Applications > Programming > Geany)
   2. Click the “New” button at the top to get a blank page
   3. Type C++ code in to the opened file:
   4. Create a new folder on the Desktop (right-click, just like in Windows)
   5. Save the file into your newly created folder as {yourfilename}.cc

2. Open a special program called a terminal and navigate to the directory where you saved your file:
   1. Go to the Applications menu (top of the screen)
   2. Click on the System Tools menu item
   3. Click on Terminal
   4. type pwd in to the terminal window ... now type ls in to the terminal window
   5. Inside the terminal type cd ~ \Desktop\{nameofyourfolder}
   6. type pwd, now type ls

3. Compile the program:
   1. In the terminal type: g++ {yourfilename}.cc
   2. Assuming our code had no errors, this will create a file called a.out
   3. In the terminal type: a.out ... this runs the program
A Terminal

A terminal window is much like a command window in Microsoft Windows. It is where we can go to navigate to any directory on our computer and type in text commands to make the computer execute certain programs.

Terminal Commands

- cd
- ls
- pwd
- using tab completion
- g++
Our First C++ Program

The ”Hello World” Program: Classic first program for beginners. All it does is print the words ”Hello World” to the screen.

HelloWorld.cc

#include <iostream>
using namespace std;
int main()
{
  cout << ”Hello World!\n”;  
  return 0;
}

NOTE: The placement of semi-colons is important. Also, C++ is case-sensitive.
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together
The Structure of C++ Programs

- Every C++ program has functions, which are collections of commands that tell the computer to do something.
- The function that begins any C++ program is called *main*. This is the first function that is called when any C++ program begins.
The Structure of C++ Programs

Line-by-Line breakdown

- `#include <iostream>`: tells the compiler to get access to many different built-in functions. For example, `cout`.
- `using namespace std`: another command we will always include that tells the compiler where to look for certain functions.
- `int main()`: tells the compiler that there is a function called `main`.
- The curly braces (`{` and `}`) signify the beginning and end of functions and other blocks of code (in this case `main`).
- `cout` is used to display text. We use the symbol `<<`, called an insertion operator, to indicate what should be printed to the screen. The quotes tell the compiler that you want to print the literal string of characters “Hello World!”
The Structure of C++ Programs

Line-by-Line breakdown - Cont.

- The `\n` stands for newline. It moves the cursor on your screen to the next line.
- Note the semicolons (;) they are added to the end of all statements in C++.
- `return 0 ;` will return the value of 0. Indicates the end of the program.
- Again, the final brace closes the main function.
Comments

Using Comments

- You can tell the compiler that a section of text is a comment
- Comments are ignored when running code
- Allows you to use any text you want to describe the real code
- You can use either //, which tells the compiler that the rest of the line is a comment, or /* and */ to block off everything between them as a comment.

Examples

- //this is a comment
- return 0; //comment
- /* All these lines will be ignored when the code is compiled */
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together
Getting and Storing Information From the User – Variables

- Programs can also accept input while they are running.
- The function we use is called cin, which is followed by `>>`, the stream insertion operator.
- BUT, before you try to receive input, you must have a place to store that input. Values are stored in variables.
- Each variable has a type. The type of the variable determines what kind of value it will store (int, char, double, etc.).
Declaring Variables

- To declare a variable, type `<variable type> <variable name>;
- Variable names can contain capital and lowercase letters, numbers, and the underscore (_)
- You cannot start a variable name with a number

Note 1:
Don’t forget the semi-colon (;)

Note 2:
Variable names are case sensitive – int MyVariable; vs int myVariable;
Examples: Declaring Variables

Examples
- int x;
- int a1, b2, c3, d4;
- char letter;
- float the_float;
//UserInput.cc
#include <iostream>
using namespace std;
int main
{
    int number;
    cout << "Please enter a number: ";
    cin >> number;
    cout << "You entered: " << number << "\n";
    return 0;
}
What Else Can We Do With C++?

What’s different from last time?

Things we haven’t seen yet.

- Notice that when you’re printing out the value of a variable, quotation marks are not used. If you used them, the program would print You entered: number.

- The lack of quotation marks informs the compiler that the program should check the value of the variable in order to replace the variable name with the value when doing output to the screen.

- Don’t worry that we’re using two insertion operators (<<, remember?) in the same line. You need to separate literal strings and variables like this.

What if the user types in something that isn’t an integer (whole number)? Run the program again. Try giving it an input that’s a character or floating-point number (decimal number).
What else can we do with variables?

- Operators give us the ability to manipulate variables and do many useful things with them.

**Examples of Operators**

- * (Multiplication)
- - (Subtraction)
- + (Addition)
- = (assigns the variable of the left side of the equal sign the value of whatever’s on the right side)
- == (checks if left value equals right value)
- > (checks if left value greater than right value)
- < (checks if left value less than right value)
- >=, <=, !=
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together

Dr. Daniel A. Ray  MCS Dept  UVa-Wise  () A Crash Course in Programming with C++  
June 8, 2010
Conditional Statements

Conditional Statements

Conditional statements allow us to program code so that some code gets executed, some code remains unexecuted and some code repeats. The decision about what code does what is based on the values of our variables. The statements that compare variables are called conditions.

The if Statement

If statements, like all conditional statements, allow the flow of a program to be changed, allowing us to write more interesting and useful code.

The Structure of an if Statement

if (condition here is true) {
    Execute all statements between the braces ( {... } )
    Otherwise skip to the next line after the closing brace ( } )
}
this line is executed whether or not the condition is true
The *else* Statement

Used in conjunction with the if statement, the code after it is executed if the statement being checked in the if statement is false.

**The Structure of an if-else Statement**

```c++
if (condition here is true) {
    Execute all statements between the if statement’s braces
}
else {
    Execute all statements between the else statement’s braces
}
continue executing the rest of the code after that
```
Our Next Program

MultipleOptions.cc

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

int main
{
    int age; // Declare a variable
    cout << "Please input your age: "; // Asks for age
    cin >> age; // The input is put in age
    if (age < 100) // If the age is less than 100
    {
        cout << "You are pretty young!\n"; // Just to show you it works
    }
    else if (age == 100) // use else
    {
        cout << "You are old\n"; // Just to show you it works
    }
    else //only if neither of the first two is true
    {
        cout << "You are really old\n"; // Executed if both fail
    }
    return 0;
}
```
Logical Operators

What if you want to check more than one condition at once? For instance, checking to see if sex==’m’ and age>65.

Logical Operators

- NOT (!) - accepts one variable. If that input is true, it returns false; if the input is false, it returns true;

- AND (&&) - accepts two variables (one on each side). It returns true if BOTH inputs are true. If one or both inputs are false, it returns false;

- OR (||) - accepts two variables (one on each side). It returns true if one or both inputs are true and returns false if both inputs are false;
Assignment #1

Re-write the code in MultipleOptions.cc. The new code will determine what range the age is in, and print out the proper response, as shown in the table below. You may only use if statements (no else statements), and you will also need to make use of logical operators.

Response Table

<table>
<thead>
<tr>
<th>Age Range</th>
<th>Printed Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-12</td>
<td>You are a child</td>
</tr>
<tr>
<td>13-19</td>
<td>You are a teenager</td>
</tr>
<tr>
<td>20-35</td>
<td>You are a young adult</td>
</tr>
<tr>
<td>36-64</td>
<td>You are an adult</td>
</tr>
<tr>
<td>&gt; 65</td>
<td>You are a senior citizen</td>
</tr>
</tbody>
</table>
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together
Loops

Making Things Happen Again, and Again, and Again, and Again, and Again, and Again

- Loops allow certain blocks of code to execute more than once, until some pre-defined condition is met
- There are three types of loops:
  - *while* loops
  - *do..while* loops
  - *for* loops
- Each has its particular uses, but we will focus on *for* loops, which are the most flexible
Syntax of *for* Loops

Basic Syntax

```cpp
for (variable initialization; condition; variable update ){
    If condition is true...
    Execute all statements between the braces ({} )
    Execute the variable update statement
    Check to see if condition is still true
}
```

this line is executed next and then the code continues on
The 3 Parts of the *for* Loop Header

- **variable initialization** – allows you to either declare a variable and give it a value or assign a value to an already existing variable
- **condition** – same as with if statements, loop will continue to iterate while the condition is true
- **variable update** – easiest way for a for loop to handle changing variable in the condition
  - $x++$ (increments the value of $x$ by one)
  - $x = x + 5$ (sets the value of $x$ to itself plus 5)

**Question:**

Is it possible to construct a loop that will not stop looping?
A Simple \textit{for} Loop in Action

**FirstForLoop.cc**

```cpp
#include <iostream>
using namespace std;
int main
{
    //The loop repeats as long as \( x < 10 \). \( x \) increases by one every loop
    for (int x = 0; x < 10; x++)
    {
        cout << x << endl;
    }
    /* Keep in mind that the loop condition checks
    the condition statement before it loops again.
    Consequently, when \( x \) equals 10 the loop breaks.
    \( x \) is updated before the condition is checked. */
    cout << x << endl;
    return 0;
}
```
Assignment #2

http://spirit.mcs.uvawise.edu/C++Assignments/assign2.pdf

Things to Consider

- How would you do this for one item?
- How would you do this for all the items, but with no loops in your code? (You’ll need if statements probably)
- Now, what part of your loop-less code keeps repeating? How many times does it repeat?
One Last Look at Loops

Before we move on to the next thing, let’s pause to take a look at what are called “nested loops”

NestedLoop.cc

```cpp
#include <iostream>
using namespace std;
int main
{
    for (int x = 0; x < 10; x++)
    {
        for (int y = 2; y < 5; y=y+2)
        {
            cout << x << endl;
        }
    }
    return 0;
}
```
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together
Functions

Thus far, all of our programs have had only one function, called “main”.

It is possible (and advisable) to divide our code into multiple independent functions.

Functions are blocks of code that perform a number of pre-defined commands to accomplish something productive.

Functions that a programmer writes generally require a prototype. Like a blueprint, the prototype tells the compiler what type of value the function will return, as well as what type(s) of argument(s) (if any) the function accepts as input.

Not all functions are written by you (the programmer). Some functions come pre-built with C++. (We will see more on this later.)
Manageable “Chunks” of Code

Example of a Programmer-Built Function

FunctionExample.cc

```cpp
#include <iostream>
using namespace std;
int mult (int x, int y); //prototype, don’t forget semicolon
int main
{
    int a;
    int b;
    cout << "Please input two numbers to be multiplied: ";
    cin >> a >> b;
    cout << "The product of your two numbers is " << mult (a, b) << "\n";
    return 0;
}

//the function is given down here
int mult (int x, int y)
{
    int product = 0;
    for(int count = y; y > 0; y = y - 1){
        product = product + x;
    }
}
```

Dr. Daniel A. Ray  MCS Dept  UVa-Wise  ( ) A Crash Course in Programming with C++  a  June 8, 2010  36 / 47
General Rules for Functions

- Functions are declared by prototypes given above main
- Functions are defined and given a block of code (usually after main, but never inside main)
- When the programmer actually defines a function, it will begin with the header, (the prototype minus the semicolon)
- There should be a block of code ({···}) that the function is to execute, just like with main()
- Any of the arguments (variable values) passed to the function can be used as if they were declared in the block
Other Important Things to Remember

- Remember what kind of variables are being passed to and from the function
  - I can do this.: int temp = mult(5,4);
  - Can I do this?: char temp = mult(5,4);
  - How about this?: int temp = mult(5.5, 4.3);
- Does this make sense?: cout << mult(5,4);
Why Do We Even Worry About Functions?

- For example, a programmer may have a block of code that is repeated forty times throughout a single program. A function to execute that code would save a great deal of space and time and would make the program more readable.

- Having one copy of the code makes it easier to make changes. Would you rather make forty small changes scattered all throughout a potentially large program or just make one change to the function body?

- Functions allow us to break programs down into logical parts
Using Pre-Defined Function?

- Some functions are already pre-defined, and we can get access to them without writing them ourselves if we include the write file (remember: `#include `<iostream>`)
- For instance: a function already exists which allows us to get random numbers
- Include the line `#include `<cstdlib>` at the top of our code
- Now, whenever I write `rand()` % `{somenumber}` I get a random number between 0 and one minus whatever number `{somenumber}` is.
Some functions are already pre-defined, and we can get access to them without writing them ourselves if we include the write file (remember: `#include <iostream>`) 

For instance: a function already exists which allows us to get random numbers 

Include the line `#include <cstdlib>` at the top of our code 

Now, whenever I write `rand() % {somenumber}` I get a random number between 0 and one minus whatever number `{somenumber}` is. 

```
int a = rand()%13;
```
Assignment #3

http://spirit.mcs.uvawise.edu/C++Assignments/assign3.pdf
http://spirit.mcs.uvawise.edu/C++Assignments/assign3Main.cc

Things to Consider

- Don’t change the main function
- Consider how you know what the functions should look like
Outline

1. What is C++?
2. Our First C++ Program
4. What Else Can We Do With C++?
5. Building Programs that Decide What to Do
6. Making Things Repeat
7. Manageable “Chunks” of Code
8. Grouping Variables Together
Consider having to store all of the grades for students in a particular class. (Assume it’s a large class)
Consider having to store all of the grades for students in a particular class. (Assume it’s a large class) That’s potentially a lot of variables...
Grouping Variables Together

Variables, Variables Everywhere...

Consider having to store all of the grades for students in a particular class. (Assume it’s a large class) That’s potentially a lot of variables... Instead, we can use what are called arrays.

Definition

Arrays are useful structures that have one name for a group of variables of the same type. Each individual variable is accessed by a unique numerical index.
How to Use Arrays

- Declaring an array (w/100 integers): int examplearray[100];
- To access a specific element of the array, you put the array name and, in brackets, an index number; this corresponds to a specific element of the array.
  - cout<<examplearray[33];
  - int a = examplearray[42];
- **Important**: the first index number of an array is zero.
What advantages do we get with grouping elements into arrays?

- Only one name to remember
Arrays and *for* Loops

- What advantages do we get with grouping elements into arrays?
  - Only one name to remember (still a lot of numbers to remember, right?)
Arrays and *for* Loops

- What advantages do we get with grouping elements into arrays?
  - Only one name to remember (still alot of numbers to remember, right?)
  - We can use these numbers much more easily inside *for* loops.
Example of a Programmer-Built Function

ArrayExample.cc

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

double avg (int grades[]); // prototype, don't forget semicolon
void printGrades (int grades[]);

int main {
    int studentGrades[20];
    for(int count=0; count<20; count++)
    {
        cout << "Enter grade " << count << ": " << endl;
        cin >> studentGrades[count];
    }
    cout << "The grades are: " << endl;
    printGrades(studentGrades);
    cout << "The average of the grades is: "
    cout << avg(studentGrades) << endl;
    return 0;
}

double avg (int grades[])
{
    double total = 0;
    double answer = 0;
    for(count=0; count<20; count++)
    {
        total = total + grades[count];
    }
    answer = total / 20;
    return answer;
}

void printGrades (int grades[])
{
    for(count=0; count<20; count++)
    {
        cout << grades[count] << ", " << endl;
    }
}
```

Dr. Daniel A. Ray  MCS Dept  UVa-Wise  () A Crash Course in Programming with C++ a  June 8, 2010  46 / 47
Things to Consider

- Building your own `main`
- How do you swap two items in an array?
- How do you pass an array as a parameter to a function? (Prototype – `int printArray(int myArray[]);`)
- What is this “bubble sort” thing?