Programming - The *Very* Basics

Dylan Zwick

Tuesday, June 17th, 2008

It is not assumed that you have a programming background for this class, although it is assumed that you have enough of a mathematical background that picking up the basic concepts of computer programs and algorithms will not be too difficult.

I just want to stress at the outset of this talk and this class that this is most definitely not a class on computer programming or software development. It is a math class designed to teach you some algebraic geometry, and some of the computational theory behind it. As a result, I plan to teach you just the computer basics that you need to understand in order to implement the algorithms discussed in the book. It is hoped that we should be able to get these basics under our belt quickly enough that we can move on to programming the type of mathematics we've been discussing in class very soon. Anything else you need, we'll learn on the way.

That being said, please take the time to play around with this stuff and definitely ask me any questions that you have. You want to have this basic stuff down.

1 The Basic Concepts

A computer program is a set of text that a computer reads and implements. The program is usually written by a person in a computer language (we'll be using the language C++) and the computer will then take the program and translate it into a language that the computer understands.

As such, there are two fundamental and very different requirements for a working computer program. The first is that the computer must understand what the program is telling it to do. In computer jargon, this means the program must compile. A program compiling means that the computer has read the program text, and has made sense of it. If a program does not make sense to a computer, it will spit out a bunch of errors when you try to compile the program, and the computer will not be able to run the program at all until these errors are addressed.

The second requirement is that what you've told the computer to do is actually what you want the computer to do. This is much more difficult than it sounds. In fact, this is frequently the more difficult of the two requirements. For example, there's an old joke among computer nerds about the programmer who used a full bottle of shampoo in the shower. The problem was, he read the directions, and they said "wash, rinse, repeat". He continued to do this until he ran out of shampoo! This is a situation in which the programmer understood what the shampoo bottle was telling him to do, but what the shampoo bottle told him to do and what he was suppose to do were two very different things.

Today, we're going to play around with some very simple programs that illustrate some of the basic concepts of computer programming: the program itself, input and output, loops, and decisions. These basic concepts will be illustrated with four very simple programs, and then at the end you'll be asked to implement a more complicated program that utilizes some of the concepts you'll learn today.

2 Creating the Program

Log into a computer and open up a terminal. Create a directory for the REU using the mkdir command. So, for example, you could create a directory called "Serre" by typing:

```
mkdir Serre
```

To move into a directory, you use the cd command. So, to move into the newly created directory you wold type:

cd Serre

Pleae create a directory for this class, and then inside that create a directory for what we're going to be doing today. Then, move to the directory for today.

Once that's done create another directory called "HelloWorld" and move there. Once you're in this directory type:

emacs hello.cpp &

This opens up the text editor emacs and within emacs creates and opens the file hello.cpp. The prefix .cpp tells emacs that this will be a C++ program file, and emacs adjust accordingly. The & at the end tells the computer to run emacs in the background, allowing us to do other things with the terminal at the same time if we so desire.

In emacs, type in the following program:

```
#include <iostream>

int main()
{
  std::cout << "Hello World\n";
  return (0);
}
```

Once you have done this type the command:

g++ -g -Wall -ohello hello.cpp

This tells the computer to compile the program. The -g enables debugging, the -wall turns on all warnings, the -ohello tells it to create an executable file named hello, and hello.cpp specifies which file to use as the sourcecode. Once this is done, assuming you entered the text correctly, there should be another file in the directory named "hello". Just type:

hello

to run the program. Hello World! If you've never programmed before, congratulations! You've just written your first program!

3 Output and While Loops

I've held your hand so far, but I'm not going to be nearly as detailed from here out. If you have any questions, please ask. The next thing we're going to be learning are the basic input and output commands. Please create another program directory called "Count" and create a program file with this text:

```
/***********************
 * count.cpp           *
 * ---------           *
 * This is a program   *
 * that counts from 1  *
 * to 100 and demon-   *
 * strates the use of  *
 * the while loop.     *
 ***********************/

#include <iostream>

int main(void)
{

  int i = 0;

  std::cout <<"I'm a super duper counter. Check this out: ";

  while(i <= 100)
    {
      std::cout << i << " ";
      i++;
    }

  std::cout <<"\n I'm so awesome!\n"; \\Man, this program is vain!

  return(0);
}
```

Play around with this program for a few minutes and read through it so that you understand what it's doing. The basic commands for input and output in C++ are cin and cout. These are both pretty cool, and somewhat complicated, commands and we're not going to be learning everything about them. Just read through and make sure you understand what the program is doing, and how we're using cout here. We'll use cin in the next sample program.

Also, there's something very important in this program that wasn't in the last program. Comments! There are two ways of leaving comments in a program. You can type two forward slashes and then everything on the same line that comes after the two forward slashes is ignored by the compiler. Or you can type a forward slash and then a star. This indicates a much longer comment, and the compiler will ignore everything until it sees a star and then a forward slash. The longer comment is illustrated above, along with a shorter comment in the middle of the program.

4 Input

Frequently in computer programs you want a computer program to do something many times, and the number of times may not be predetermined, but may depend upon other data that can change from run to run. This is where loops come in. A loop is something that a computer does over and over again until some predetermined condition is achieved. We saw an example of this in the last program, where an action was performed over and over again until a counter reached 100. This final value of 100 was set by the program itself, not the user, but it was still an example of a while loop. Be careful with these, because it's easy to write a loop that never ends! I've written a program to illustrate user input and the while loop called "CountTo". Type it in and check it out:

```
#include <iostream>

int main(void)
{

  int max;
  int i = 0;

  std::cout <<"I’m a super duper counter. I can count real high. To wh

  std::cin >> max;

  std::cout <<"OK, here I go ";

  while(i <= max)
    {
      std::cout << i << " ";
      i++;
    }

  std::cout <<"\n I’m so awesome!\n";

  return(0);
}
```

5 Choices

The next and last thing that we’re going to look at is how to tell a computer program to do one thing if a given condition is satisfied, and another if another conditions is satisfied. This along with loops are the two things a computer program must be able to do, and that’s pretty much it! Anything else you want a computer to do you can pretty much accomplish by using choices and loops. Anyways, here’s an example of an “if... else” statement in a program. Check it out, run it, and make sure you understand it. If you have any questions, please ask.

```
#include <iostream>

int main(void)
{

  int first, second;

  std::cout <<"Not only can I count, I know a lot about the ordering o

  std::cout <<"Please enter the first number: ";
  std::cin >> first;

  std::cout <<"Please enter the second number: ";
  std::cin >> second;

  if(first > second)
    {
      std::cout <<"\nThe first number you entered, " << first << ", is
    }
  else if(second > first)
    {
      std::cout <<"\nThe second number you entered, " << second << ",
    }
  else
    {
      std::cout <<"\nYou entered the number " << first << " twice, sil
    }

  return(0);
}
```

6 Putting it Together

Finally, I want you to write a computer program on your own. Create a new directory, open up a .cpp file, and write a program that asks the user to input two integers, and then finds the greatest common divisor of those two integers. The greatest common divisor can be found using an ancient (in some sense, the first) algorithm called the Euclidean algorithm.

The basic idea behind the Euclidean algorithm is that you start with two numbers: a, b. You divide one number by the other and you get a remainder. You then repeat the process in the way outlined below:

$$\begin{aligned} a &= q_0 b + r_0 \\ b &= q_1 r_0 + r_1 \\ r_0 &= q_2 r_1 + r_2 \\ r_1 &= q_3 r_2 + r_3 \\ &\vdots \\ r_{n-2} &= q_n r_{n-1} + r_n \\ r_{n-1} &= q_{n+1} r_n \end{aligned}$$

where r_n is the last nonzero remainder. This r_n is the g.c.d. of the number a and b. Try to write a computer program on your own that implements this algorithm, and play around with it for a few minutes to make sure it works right. Once you've got it working, congratulations! Euclid would have been so proud.