Introduction

http://people.sc.fsu.edu/~jburkardt/isc/week01/lecture_01.pdf

.....

ISC3313:
Introduction to Scientific Computing with C++
Summer Semester 2011

.....

John Burkardt
Department of Scientific Computing
Florida State University

Last Modified: 09 May 2011

- **Introduction**
- Textbook
- Assignments
- Lab Accounts and Software
- HELLO
- ADD_INTS
- MANDEL
- Conclusion

INTRO: Programming for Scientific Applications

I expect you are in this class because you:

- don't know how to program,
- or have never programmed in C++,
- or have little experience in scientific programming in C++.

My goal is to **teach you to program**. In C++ of course. And we will use many simple scientific applications for our examples.

You will not be required to understand physics, or chemistry or biology or whatever science our example comes from. You will need to know a little bit of math, but if you know how to differentiate a simple formula, and what the integral of a function means, you'll be fine.

INTRO:

Your instructor:
  John Burkardt
  Office: 445 Dirac Science Library
  Email: jburkardt@fsu.edu

Your TA:
  Detelina Stoyanova
  Office: 422J Dirac Science Library
  Email: dks10d@fsu.edu

Lab and Office Hours (starting NEXT week):
  Detelina or I will be in the lab,
  Monday, Wednesday, and Friday, from 11:00 to 12:15.
  You can come to work on homework, or ask us questions.

INTRO:

The course website is: http://people.sc.fsu.edu/~jburkardt/isc

For each week of class, there will be:

- lecture notes,
- examples discussed in class,
- assignments (reading, in-class, homework)

Some of this information will also be on the class Blackboard site.

INTRO: C++ topics

C++ topics we will learn include:

- the format of a program;
- variable names, statement structure;
- steps in creating, compiling, and running a program;
- program control using **if/else**, **for**, **while**;
- creating and using functions;
- arrays and vectors;
- pointers;
- input/output and files.
- classes

INTRO: C++ topics

You may have heard that C++ is an object-oriented language. The topic of classes, which comes at the very end of our course, is just the beginning of the object oriented topics in the book.

I believe that, in a first course on C++, it's better to concentrate on the simple, straightforward tasks of computing, and to save the object stuff for your second class! That is why I specified the "late objects" version of the textbook.

This means that we are learning a simpler, old-fashioned kind of programming known as procedure oriented programming. I do this because I think it is the fastest, simplest way to get you writing correct and useful programs.

INTRO: Scientific Applications

Our classes will include scientific applications, including:

- solving a problem by iteration (try, try again);
- working with polynomials;
- the use of random numbers for simulation;
- the solution of a nonlinear equation;
- the approximation of a function using a few data points;
- estimating an integral;
- tracing the solution of a differential equation;
- creating scientific graphics.

INTRO: Programming

During lectures, I will present many programming examples.

Generally, each week we will have a short in-class programming exercise.

Many of your homework assignments will involve writing a program, based on material presented in the lectures and the in-class exercises.

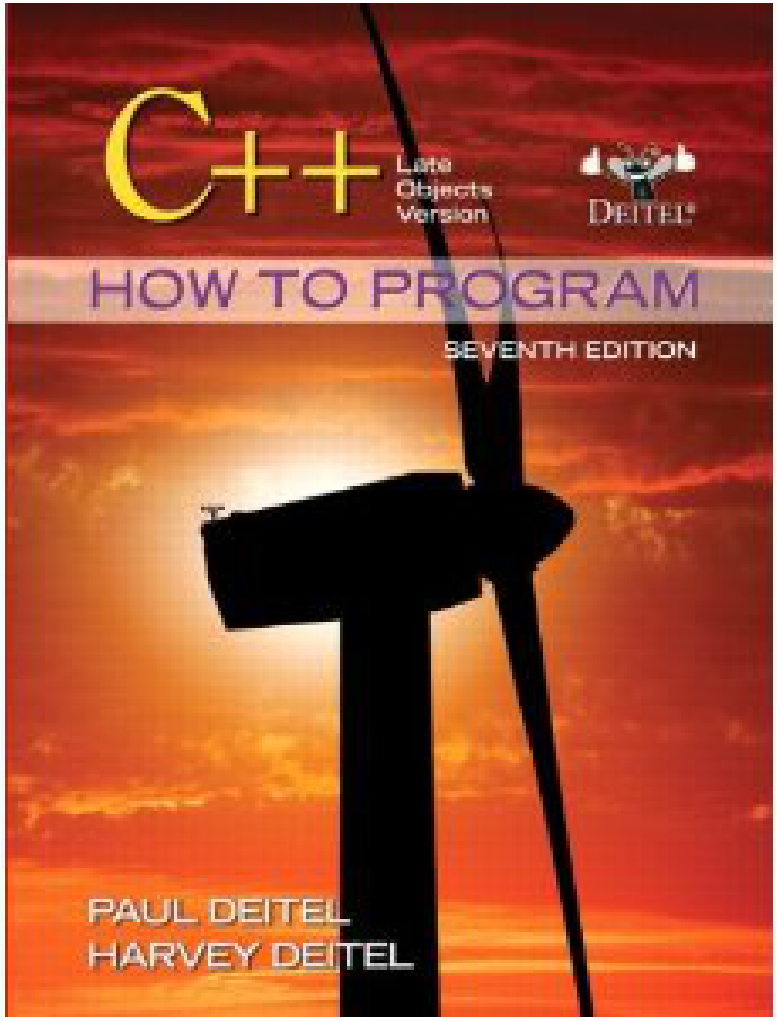
I want to expose you to programs, to get you started on simple guided exercises, and then to have you write short simple programs on your own.

Our goal is to learn to write small programs for practical scientific applications.

- Introduction
- **Textbook**
- Assignments
- Lab Accounts and Software
- HELLO
- ADD_INTS
- MANDEL
- Conclusion

TEXT: Deitel & Deitel

Our textbook is "C++: How to Program", Late Objects Version, 7th Edition, by Deitel and Deitel.



TEXT: What is the Book For?

I will expect you to:

- read Chapters 1 through 9;
- be responsible for (selected) material from these chapters;
- to do homework exercises I choose from the book.

Buying the book entitles you to access to a textbook website which includes video notes and example programs.

If you take a second course in C++, or a job where you need C++, you will find this textbook useful as a reference and guide.

TEXT: Deitel & Deitel

The book is not a perfect book for this course.

Sometimes, it presents a simple idea, and then tells you ways to make it more complicated. As beginning programmers, we don't want to hear about that yet!

It likes to use a system of diagrams called UML (Unified Modeling Language) to illustrate its explanations of C++ commands. I don't find UML useful, and I will give you my own explanations.

I will have other comments and complaints to make about the book as we go along. However, almost everything I want you to know is in the book, so I will mostly try to tell you what parts to ignore and when to pay close attention.

TEXT: Reading for this Week

The first chapter contains a lot of material I am not interested in. So you may skim over most of it. But please read and understand the following sections:

- 1.7 Machine Language, Assembly Language, High Level Language
- 1.8 History of C and C++
- 1.14 Typical C++ Development Environment
- 1.16 Test-Driving a C++ Application

- Introduction
- Textbook
- **Assignments**
- Lab Accounts and Software
- HELLO
- ADD_INTS
- MANDEL
- Conclusion

ASSIGNMENTS: Lab Exercises

I plan to have a short lab exercise each week, probably on Thursday.

The lab exercise is done in class, during the last 10 or 15 minutes of class. It is designed to be a warmup exercise, introducing some program, or topic, or data needed for the homework.

You should be able to complete the exercise in class. On completing the assignment, please have Detelina check your work so you get credit. At that point, you are free to go.

ASSIGNMENTS: Programs

Most weeks, you will have a short programming homework assignment.

The programming assignment will usually involve writing or modifying a C++ program and running it.

You will have a week to do the homework assignment.

You may turn in your programming assignment by emailing it to Detelina.

ASSIGNMENTS: Late Policy for Exercises and Homework

You can turn in one lab assignment (in class exercise) and one homework program late, with no questions asked and no penalty.

After that, late assignments are penalized;

In any case, assignments more than a week late will not be accepted.

ASSIGNMENTS: Midterm

A little more than half-way through the semester we will have a midterm exam.

The midterm will assess your ability to write programs, to spot errors, and to explain some basis procedures of scientific computing.

The midterm exam will be in-class.

ASSIGNMENTS: Project

The FSU Computer Competency Component of this class requires that each student prepare and present a project.

After the midterm exam, we will discuss suitable project topics, how the project will be written up, and presented.

The project presentations will take place during the final week of class.

- Introduction
- Textbook
- Assignments
- **Lab Accounts and Software**
- HELLO
- ADD_INTS
- MANDEL
- Conclusion

LAB: Class accounts

Each student registered for this class is authorized to log into the lab computers, as well as the hallway computers on the 4th floor of Dirac (the main floor for the Department of Scientific Computing).

You log in using your FSU ID and password.

If you are unable to log in, please speak to Detelina or send her an email at **dks10d@fsu.edu** so she can look into the problem.

All these computers share a single file system. You can log into any one of these computers and see your files.

You can access your files from a remote computer, using the **ssh** or **sftp** programs on Unix, or the **putty** program on Windows.

From your machine, you want to reach the address **pamd.sc.fsu.edu**.

LAB: Use of Computers

During lectures, we may illustrate a concept by creating and using a program. You are welcome to try to follow along on your computer.

You may download the lecture notes from the class web site and view them on your computer rather than on the projection screen. The class web site is **http://people.sc.fsu.edu/~jburkardt/isc**

During class lab sessions, we will have exercises that you must complete in the lab.

Your homework will include programming exercises, which can be done in the lab or your own computer.

LAB: Open Lab Time

While you can do your programming homework anywhere, you may want to do it here in the classroom.

The classroom is normally locked, and not available. However, either Detelina or I will be here in the classroom, Monday, Wednesday and Friday, from 11:00 to 12:15, starting next week.

You are free to come to the lab and work then, to ask us for help with assignments, or to talk about C++ programming, setting up your computer, and so on.

If these combined office hours/lab hours are not convenient for you, please let me know and we can work out other times or places.

LAB: Politeness

Please remember that you are sitting in a class, with other students next to you, and an instructor in front of you.

I said you are encouraged to follow along on the computer while I am lecturing, and that means you can be looking at class notes, or trying out an example, or checking on the web for help with a topic, or even, (quietly, please!), trying to catch up on your homework.

But playing computer games, watching movies, chatting with friends during lecture time is inappropriate and distracting. If I find this becoming a problem, I will disable the computers.

LAB: Software

Some software programs are installed on the lab machines.

- **kedit** and **gedit** are simple text editors.
- **netbeans** is an intelligent program editors;
- **g++** is the Gnu C++ compiler;
- **gnuplot** is an interactive graphics program;
- **xmgrace** is an interactive graphics program;
- **dislin** is a graphics library;
- **eog** is a program that can display graphics files;

On Thursday, we will learn a little about **kedit** and **g++**!

LAB: Software for Your Own Computer

If you plan to work on your own computer, you may be able to install many of the programs available in the lab.

You probably already have a favorite editor on your machine, so don't worry about copying **kedit**.

You may find the free programming environments **eclipse** or **netbeans** useful as an "integrated development environment" (IDE). We will talk about IDE's later in the class.

If you are using a PC, you might check into **Visual C++ Express**, a free C++ compiler from Microsoft.

It will be helpful to be able to have some kind of scientific graphics program on your machine. **gnuplot**, **xmgrace** and **dislin** are all free, but you can wait til we try them out during a lab session to see if you like them.

- Introduction
- Textbook
- Assignments
- Lab Accounts and Software
- **HELLO**
- ADD_INTS
- MANDEL
- Conclusion

HELLO

Since this is the very first class, you can't possibly know anything about C++.

But before we even learn the rules of C++, let's just see what a C++ program looks like, make some guesses about the rules, and see how such programs can be used.

When you write a program, you are writing it in what is called a *high level language*. Your program is a text file, sometimes called the source code. The name of a C++ text file usually includes a *file extension* or "last name" so that it can be recognized.

I will try to use the last name **.cpp** for my source codes.

HELLO: Source Code

Here is your first C++ program, called **hello.cpp**

```
# include <cstdlib>
# include <iostream>

using namespace std;

int main ( )
// This program will say hello.
{
  cout << "Hello, world!" << endl;
  return 0;
}
```

HELLO: A few comments

Notice that the entire program seems to be in lower case (with two exceptions).

This is typical of C++ programs. Later, we may see some names spelled in ALL CAPITALS; this is done to indicate that the name has a special property.

C++ is case sensitive. The words **long**, **Long** and **LONG** are different words to C++.

C++ uses punctuation marks in a particular way:

- Statements, which carry out an action, end with a semicolon ;
- Groups of statements are collected by curly brackets { and }
- Comments begin with a double slash //;
- Lines beginning with # are preprocessor directives.

HELLO: Making the program run

In order to convert my source code into a program I can run, I need to use a compiler. We will be using the Gnu C++ compiler. To compile the hello program, we would type:

```
g++ hello.cpp
```

The first thing the compiler does is a *preprocessing step*, where it copies information requested by my **include** statements.

Then it checks the text of my program to make sure it is correct. Every line of the program must follow the rules for C++ programming. Especially when you are beginning to program, you may spend some time trying to get past this stage!

HELLO_OOPS

Here is a slightly defective version of the program, called **hello_oops.cpp**:

```
 1  # include <cstdlib>
 2  # include <iostream>
 3
 4  using namespace std;
 5
 6  int main ( }
 7  {
 8    cout << 'Hello, world!" << endl;
 9    Return;
10  }
```

HELLO_OOPS

Here are the error messages I got when I made some mistakes in my program:

```
hello_oops.cpp:8:11:
  warning: missing terminating ' character
hello_oops.cpp:8:
  error: missing terminating ' character
hello_oops.cpp:6:
  error: expected primary-expression before } token
hello_oops.cpp:6:
  error: expected , or ; before } token
hello_oops.cpp:6:
  error: expected declaration before } token
```

The compiler seems to complain about some errors several times, and it also missed one error.

HELLO: Making the program run

Once we correct the language mistakes and the compiler accepts our source code, it creates a new file to hold the executable program. This file usually has the name **a.out**.

(It's easy to change this name to something friendlier. But for now, we'll go with the default name!)

To actually run the program, we type

```
./a.out
```

and for our example, we'd see *Hello, world!*

HELLO: Remarks

Perhaps the most surprising thing about this program is that it is trying to do something very simple (say hello), and yet only one line of text seems to be doing that. What are the other lines of text doing?

To start with, let's ignore the first three lines.

The line **int main ( )** starts what C++ calls a function. A function is a group of statements organized to carry out a specific computation.

The function has a name, can accept input, and can produce output, which is often a result.

Every C++ program must contain a **main** function, which is where the program begins to run. The main function can call other functions to help it.

HELLO: Remarks

The name of this function is **main**.

If the function expected input, it would be supplied between the parentheses, but we see they are empty: **main()**.

The curly brackets { and } which follow enclose all the statements that make up the function.

The **return** statement reports the value of the function. In C++, a main function returns 0 if it worked as expected, or 1 (or some other nonzero value) if a problem was encountered.

The **int** that is part of **int main ( )** tells us that the kind of result this function returns will be an integer, not a real number.

HELLO: Remarks

The "interesting" line in hello.cpp says "Hello, world!" to us.

```
cout << "Hello, world!" << endl;
```

The word **cout** represents output to the console, or the user's computer screen. The operator $<<$ indicates that the following item is to be printed; in this case, the string "Hello, world!".

Notice that the quotes do not appear on the output. They group the characters into a single string.

If I actually wanted quotes that appear in the output, I use the combined symbol \":

```
cout << " \" Hello, world! \" " << endl;
```

HELLO: Remarks

What does the second operator << doing with **endl**?

```
cout << "Hello, world!" << endl;
```

Why don't we see the letters **endl** on the screen?

If we (carefully) rewrite the line as

```
cout << "Hello, world!";
```

we see that **endl** is a way of starting a new line of output.

By the way, another way of getting a carriage return or new line is to use the symbol **n**. It means the same thing as **endl**. In fact, I am used to using that symbol instead, so I usually write

```
cout << "Hello, world!\n";
```

HELLO: Those Two First Lines

Let's go back and consider those first two lines now:

```
# include <cstdlib>
# include <iostream>
```

Most C++ programs need to take advantage of built-in names and functions. However, C++ requires that you explicitly ask for these functions. You do this with preprocessor commands, sometimes called "include statements". Preprocessor commands begin with the sharp or pound sign #, and tell the compiler to add the necessary definitions to your program.

The C standard library or **cstdlib** is typically needed by most programs, and I will *always* include it in mine.

If your program does any printing or input/output, you must tell the compiler to include the **iostream** information.

HELLO: The Third Line

The third line is

```
using namespace std;
```

In order to avoid confusion, when C++ provides special names and functions to you, it includes a prefix that identifies the library from which it came. In particular, the full name of **cout** is **std::cout** and **endl** is really **std::end**. However, it is inconvenient to use these longer names, so we say

```
using namespace std;
```

to indicate that we want to use the shorter names.

I will always use this convention.

- Introduction
- Textbook
- Assignments
- Lab Accounts and Software
- HELLO
- **ADD_INTS**
- MANDEL
- Conclusion

ADD_INTS: Numerical Calculations

We usually imagine computers evaluating formulas and printing results. The HELLO program didn't show us what that was like.

A C++ program stores numerical data in **variables**. Variables have names, and types (such as integer or real). Variables can be combined in formulas, and the value of the formula can be stored in a new variable (or it can update the value of a variable.)

In C++, an expression like

```
x = y + z;
```

does not mean that x is the same as y plus z. Instead, it means that the variable called x is to be assigned the value of y plus z.

This means that in C++ it is legal to say:

```
x = x + 1;
```

which would not make sense in algebra!

ADD_INTS.CPP

The source code is called **add_ints.cpp**:

```
# include <iostream>
int main ( )
{
  int number1, number2, number3;

  std::cout << "Enter first integer: ";
  std::cin >> number1;
  std::cout << "Enter second integer: ";
  std::cin >> number2;

  number3 = number1 + number2;

  std::cout << "The sum of " << number1
            << " and " << number2
            << " is " << number3 << std::endl;
}
```

- Introduction
- Textbook
- Assignments
- Lab Accounts and Software
- HELLO
- ADD_INTS
- **MANDEL**
- Conclusion

MANDEL

Most of the C++ programs I write, and that you will find in the book, have been written with spacing and indentation, so that it is easy to understand what is going on.

If a program is readable, you can sometimes work out what it is doing, and figure out how to fix it or improve it.

On the other hand, an unreadable program is very hard to work with.

Readable programs don't just happen. You have to look at programs and see how some writers tried to make your life easier...and some did not.

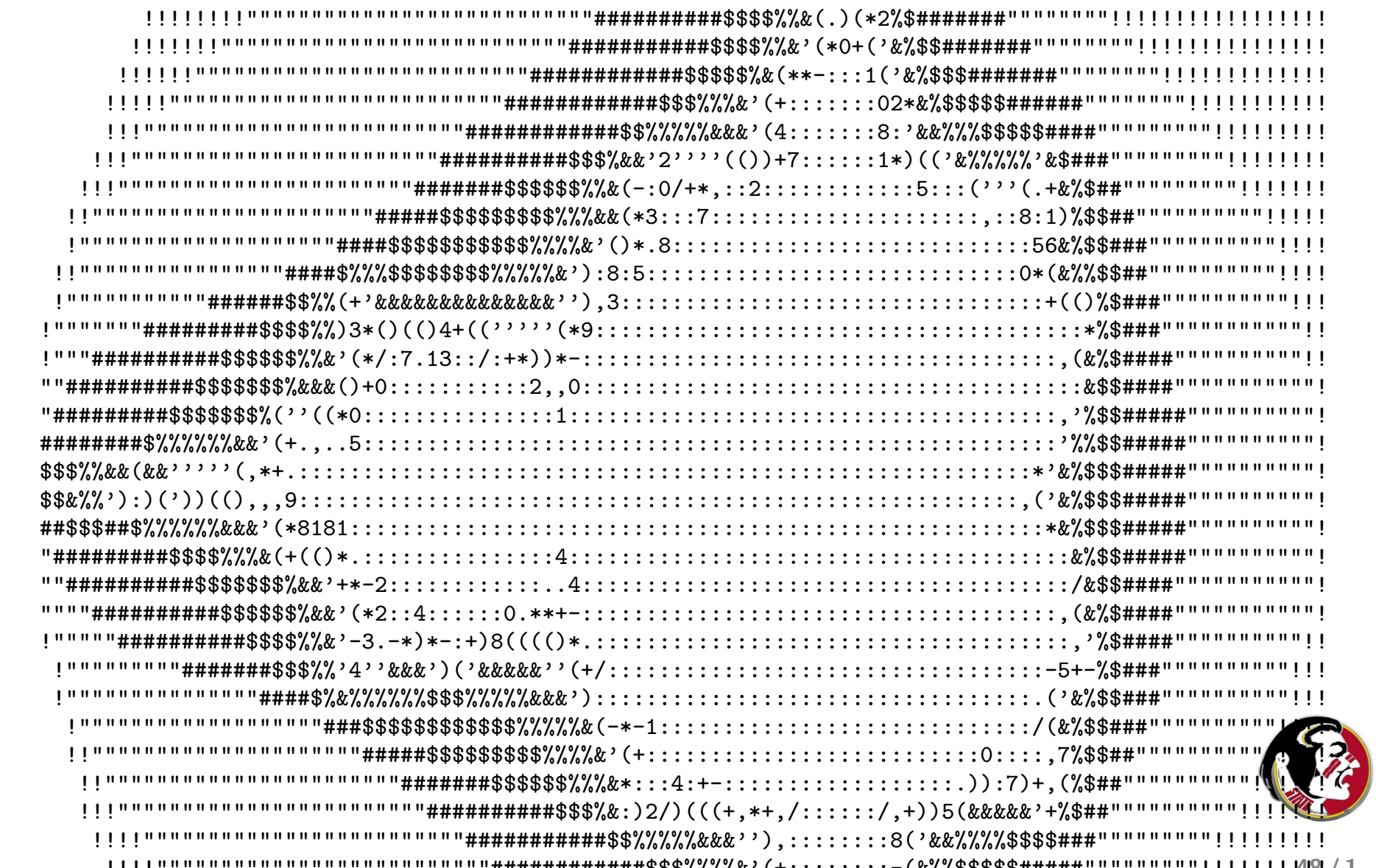
MANDEL_TXT.CPP: The Source Code

This is a legal C++ program, **mandel_txt.cpp**. From the little bit I've told you, you can at least recognize a preprocessor line, and the beginning and end of the **main** function, but believe me, even an experienced programmer would have trouble figuring out what this program is doing!

```
#include <iostream>
main(){float r,i,R,I,b;int n;for(i=-1;i<1;i+=.06,
puts(""))for(r=-2;I=i,(R=r)<1;r+=.03,putchar(n+31))
for(n=0;b=I*I,26>n++&&R*R+b<4;I=2*R*I+i,R=R*R-b+r);}
```

But let's go ahead and run it!

MANDEL.CPP: The Source Code



MANDEL: Better Graphics?

So at least the program is doing something useful: sketching the shape of the Mandelbrot set. But it really did not have to be so hard to read!

But another thing should bother you about this program, namely, the primitive output. It does show you the Mandelbrot set, but it's not very pretty.

There are many graphics formats available now that are almost as simple to use as the typewriter graphics in the previous image.

One graphics format is called PPM or Portable Pixel Map. It allows you to describe an image as though it were a piece of graph paper. At each box in the graph paper, you specify the color.

MANDEL: Better Graphics?

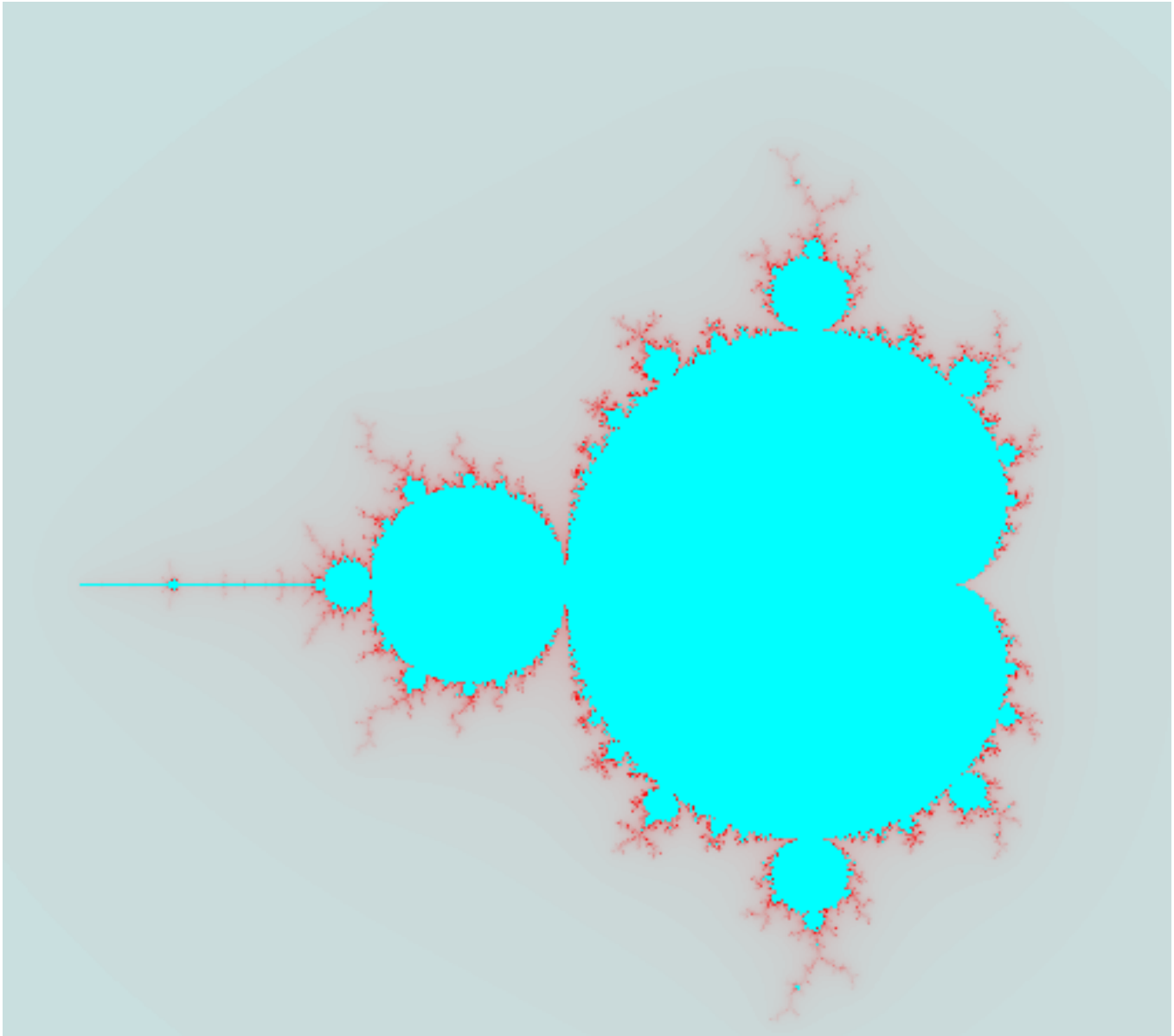
Here is another picture of the Mandelbrot set, made by a different C++ program, called **mandel_ppm.cpp**, using the PPM format for output.

By the way, if you try to run this program on your computer, you need to send the output of the program to a file, using the Unix symbol >:

```
g++ mandel_ppm.cpp
./a.out > mandel.ppm
eog mandel.ppm
```

Here, **eog** is a Gnu program which can display most graphics files.

MANDEL: PPM Graphics Output



- Introduction
- Textbook
- Assignments
- Lab Accounts and Software
- HELLO
- ADD_INTS
- MANDEL
- **Conclusion**

Conclusion

I hope you have learned a little bit about the structure of a C++ program:

- there's some initial stuff and then a main program;
- the main program contains a list of statements;
- there are input and output statements;
- there are variables to hold numbers and calculate with them.

Conclusion

I hope you have learned a little bit about how a C++ program is used:

- an editor can be used to type in a program;
- the compiler checks for language mistakes;
- the compiler turns a correct source code into an executable;
- the executable can be executed by typing **./a.out** ;
- the output from a program, which normally goes to the screen, can be sent instead to a file using the $>$ operator.

Conclusion

The examples programs I showed you today also have a moral:

- talking to the user (input/output) is important;
- numerical calculations are done with formulas that look somewhat similar to algebra;
- graphics can be created by a program as part of the calculation.

Conclusion

Read: Sections 1.7, 1.8, 1.14, 1.16 of Deitel and Deitel.

Next Class:

- An introduction to UNIX;
- setting up your directory and moving around in it;
- copying files from the browser to your directory;
- compiling, and running a program someone else wrote;
- using an editor to create your own program.

Thursday's class will include a short in-class lab exercise for credit.