Using Stoustrup's *Programming Practices and Principles in C++* as an Introductory Text for Computing I

Geoff Hagopian

College of the Desert

ghagopian@collegeofthedesert.edu

January 3, 2015

Overview

1. About Me
2. About Me
3. About Me
4. About Me
5. About Me
6. About Me
7. About Me
8. About Me
9. Stroustrup's Philosophy
10. Stroustrup's Philosophy
11. Stroustrup's Philosophy
12. Stroustrup's Philosophy
13. Stroustrup's Philosophy
14. My First Implementation

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

(II) Worked briefly in industry (Los Angeles/Hollywood).

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

(II) Worked briefly in industry (Los Angeles/Hollywood).

(III) Taught math at Community Colleges (PCC, GCC, Palomar, COD).

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

(II) Worked briefly in industry (Los Angeles/Hollywood).

(III) Taught math at Community Colleges (PCC, GCC, Palomar, COD).

(IV) Started teaching Physics 5 (Scientific Computing) in 1997.

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

(II) Worked briefly in industry (Los Angeles/Hollywood).

(III) Taught math at Community Colleges (PCC, GCC, Palomar, COD).

(IV) Started teaching Physics 5 (Scientific Computing) in 1997.

(V) Two one-year sabbaticals on cryptography and scientific visualization.

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

(II) Worked briefly in industry (Los Angeles/Hollywood).

(III) Taught math at Community Colleges (PCC, GCC, Palomar, COD).

(IV) Started teaching Physics 5 (Scientific Computing) in 1997.

(V) Two one-year sabbaticals on cryptography and scientific visualization.

(VI) Studied Racket with Steven Bloch.

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

(II) Worked briefly in industry (Los Angeles/Hollywood).

(III) Taught math at Community Colleges (PCC, GCC, Palomar, COD).

(IV) Started teaching Physics 5 (Scientific Computing) in 1997.

(V) Two one-year sabbaticals on cryptography and scientific visualization.

(VI) Studied Racket with Steven Bloch.

(VII) Built CS program at COD.

My Brief CV

(I) Earned MA in Mathematics from UC Davis.

(II) Worked briefly in industry (Los Angeles/Hollywood).

(III) Taught math at Community Colleges (PCC, GCC, Palomar, COD).

(IV) Started teaching Physics 5 (Scientific Computing) in 1997.

(V) Two one-year sabbaticals on cryptography and scientific visualization.

(VI) Studied Racket with Steven Bloch.

(VII) Built CS program at COD.

(VIII) http://geofhagopian.net/

Stroustrup's Philosophy

(A summary of S's article, Programming in an undergraduate CS curriculum)

(I) Programming is a means of making ideas into reality using computers.

Stroustrup's Philosophy

(A summary of S's article, Programming in an undergraduate CS curriculum)

(I) Programming is a means of making ideas into reality using computers.

(II) What universities produce $\neq$ what industry needs.

Stroustrup's Philosophy

(A summary of S's article, Programming in an undergraduate CS curriculum)

(I) Programming is a means of making ideas into reality using computers.

(II) What universities produce $\neq$ what industry needs.

(III) CS must emphasize software development (even at the expense of algorithmic complexity, data structures and...subsurface luminosity).

Stroustrup's Philosophy

(A summary of S's article, Programming in an undergraduate CS curriculum)

(I) Programming is a means of making ideas into reality using computers.

(II) What universities produce $\neq$ what industry needs.

(III) CS must emphasize software development (even at the expense of algorithmic complexity, data structures and...subsurface luminosity).

(IV) Fashions come and go so rapidly that only a solid grasp of the fundamentals of CS and software development has lasting value. Industry wants software "developers" more than computer scientists and engineers.

Stroustrup's Philosophy

(A summary of S's article, Programming in an undergraduate CS curriculum)

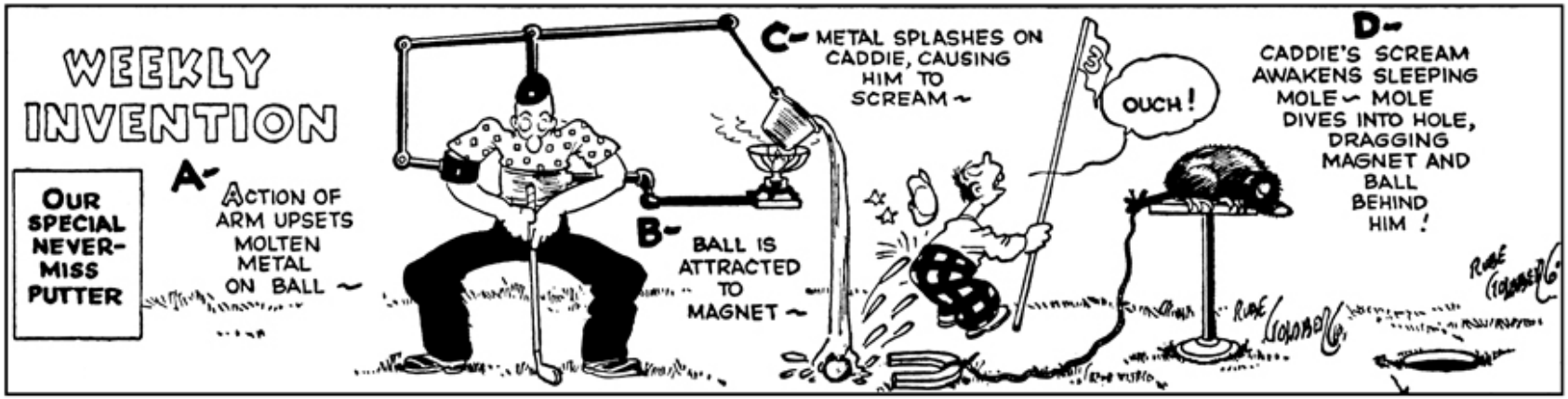
(I) Programming is a means of making ideas into reality using computers.

(II) What universities produce $\neq$ what industry needs.

(III) CS must emphasize software development (even at the expense of algorithmic complexity, data structures and...subsurface luminosity).

(IV) Fashions come and go so rapidly that only a solid grasp of the fundamentals of CS and software development has lasting value. Industry wants software "developers" more than computer scientists and engineers.

(V) Preferably, an understanding of programming extends to several kinds of languages (declarative, scripting, machine level) and applications (embedded systems, text manipulation, small commercial application, scientific computation); language bigots do not make good professionals.



For many, "programming" has become a strange combination of unprincipled hacking and invoking other people's libraries (with only the vaguest idea of what's going on). The notions of "maintenance" and "code quality" are at best purely academic. Consequently, many in industry despair over the difficulty of finding graduates who understand "systems" and "can architect software."

The study of "software" includes

- Software engineering on a small group scale, where a team collaborates on different parts of a project–this practice lays the groundwork for scaling up working on programs with millions of lines of code.

The study of "software" includes

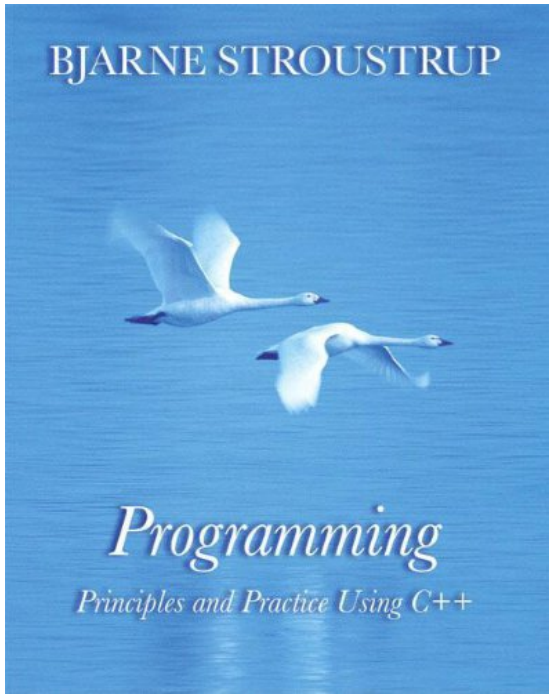
- Software engineering on a small group scale, where a team collaborates on different parts of a project–this practice lays the groundwork for scaling up working on programs with millions of lines of code.
- The use of and comparison of programming languages.

The study of "software" includes

- Software engineering on a small group scale, where a team collaborates on different parts of a project–this practice lays the groundwork for scaling up working on programs with millions of lines of code.
- The use of and comparison of programming languages.
- Individual and group projects done at each level starting in the first programming course and repeated with increasing difficulty in every software course. These projects are central to teaching the beginnings of the "higher level" project management and software engineering skills. This is where tools such as test frame-works and source control systems find their natural homes.

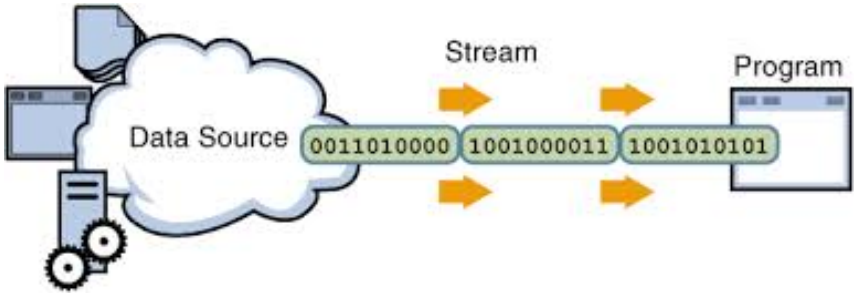
Stroustrup's Freshman Programming Class

The freshman programming class at TAMU is based on Stroustrup's *Programming – Principles and Practice using C++*. Its preface, table of contents, lecture slides, and other supporting materials can be found at http://www.stroustrup.com/Programming.



The approach is "depth first" in the sense that it quickly moves through a series of basic techniques, concepts, and language supports before broadening out for a more complete understanding. The first 11 chapters (which Stroustrup does in about 6 weeks–but I took 15) cover objects, types and values, computation, debugging, error handling, the development of a "significant program" (a desk calculator) and its completion through redesign, extension of functionality, and testing. Language-technical aspects include the design of functions and classes.

Finally, interactive and file I/O are explained in some detail. The data types used are `bool`, `char`, `int`, `double`, `string` (a variable length sequence of characters), and `vector` (an extensible container of elements). That's "the basics." At this point, the students can (in principle) do simple computations on streams of numbers and/or strings – they are by now dazed and need a break!



Invariants, interface design, error handling, and the need to reason about code to ensure correctness are central themes of the text. These habits of mind are not easily embraced (they were not for me, at least) and students may not go willingly, but these habits are essential to a sound programming foundation. Concepts and techniques are presented through concrete examples followed by the articulation of an underlying general principle. Students often struggle grasping the importance of these principles, which are seen as "abstract," so repeated application to concrete examples is essential. The style of the concrete examples reflects the principles and can – when imitated by the students – lead to later understanding;

Habits of mind.

Outcomes and What's Next

1. At the end of this part of Stroustrup's course (the entirety of my course), successful students will have no trouble with simple final exam exercises like the one shown here.

```
// produce the sum of the integers in "data.txt"
ifstream is("data.txt");
if (!is) error("data file missing");
int sum = 0;
int count = 0;
int x;
while(is>>x) {          // read into x
        sum+=x;
        ++count;
}
cout << "the sum of " << count
        << " elements is " << sum <<"\n";
```

Outcomes and What's Next

1. At the end of this part of Stroustrup's course (the entirety of my course), successful students will have no trouble with simple final exam exercises like the one shown here.
2. The next part of P^3 (chapters 12-16) will be part of our 2nd semester course, and are fun: using the FLTK (Fast Light Tool Kit, "full tick") graphing libraries. Class hierarchies and virtual functions are introduced and the fundamentals of OOP are presented as a simple response to an obvious need.

```
// produce the sum of the integers in "data.txt"
ifstream is("data.txt");
if (!is) error("data file missing");
int sum = 0;
int count = 0;
int x;
while(is>>x) {          // read into x
        sum+=x;
        ++count;
}
cout << "the sum of " << count
        << " elements is " << sum <<"\n";
```

Difficulties in Execution

True quantitative measures of success are elusive, but complaints are common:

1. The order of topics is confusing and illogical, especially to students who have programmed before and have a firm idea of what should be taught and in which order. The ordering based on programming needs and principles rather than language features is seen as "wrong and unnatural."

Difficulties in Execution

True quantitative measures of success are elusive, but complaints are common:

1. The order of topics is confusing and illogical, especially to students who have programmed before and have a firm idea of what should be taught and in which order. The ordering based on programming needs and principles rather than language features is seen as "wrong and unnatural."
2. By contrast, novice programmers do not have a problem with the early use of standard library facilities (such as iostreams, string, and vector) and do not find the early absence of pointers and arrays strange.

Difficulties in Execution

True quantitative measures of success are elusive, but complaints are common:

1. The order of topics is confusing and illogical, especially to students who have programmed before and have a firm idea of what should be taught and in which order. The ordering based on programming needs and principles rather than language features is seen as "wrong and unnatural."
2. By contrast, novice programmers do not have a problem with the early use of standard library facilities (such as iostreams, string, and vector) and do not find the early absence of pointers and arrays strange.
3. The harping on statements of principles (to achieve correctness, maintainability, etc.) is "over our heads" and "irrelevant for programmers." The latter comment proves the need for an emphasis on professionalism. This problem can be surmounted through a close tie between concrete examples (code) and statement of principles. In particular, we (also) present examples of errors to teach the students to recognize both "silly errors" and violations of principles. The students do seem to make fewer "stupid errors."

Basic Language and Algorithmic Development

These are problems I assigned to help students find the ropes of variable types and control structures in a rich mathematical context.

1. Write a program to solve quadratic equations of the form $ax^2 + bx + c = 0$. If you don't know the quadratic formula for solving such an expression, do some research. Researching how to solve a problem is often necessary before a programmer can teach the computer how to solve it. Use doubles for the user inputs of a, b, c. There are two solutions to the equation, output both x_1 and x_2.

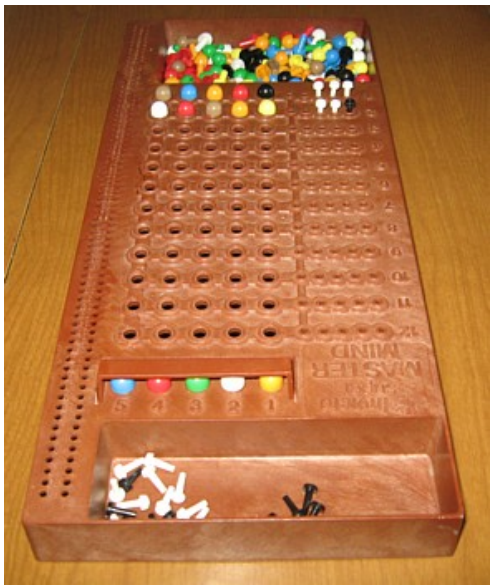
Basic Language and Algorithmic Development

These are problems I assigned to help students find the ropes of variable types and control structures in a rich mathematical context.

1. Write a program to solve quadratic equations of the form $ax^2 + bx + c = 0$. If you don't know the quadratic formula for solving such an expression, do some research. Researching how to solve a problem is often necessary before a programmer can teach the computer how to solve it. Use doubles for the user inputs of a, b, c. There are two solutions to the equation, output both x_1 and x_2.
2. Create a program to find all the prime numbers between 1 and 100. There is a classic method for doing this, called the "Sieve of Eratosthenes." If you don't know that method, get on the web and look it up. Write your program using this method. Modify the program described in the previous exercise to take an input value max and then find all prime numbers from 1 to max.

More Advanced Algorithmic Development

Implement a little guessing game called (for some obscure reason) "Bulls and Cows." The program has a vector of four different integers in the range 0 to 9 (e.g., 1234 but not 1122) and it is the user's task to discover those numbers by repeated guesses. Say the number to be guessed is 1234 and the user guesses 1359; the response should be "1 bull and 1 cow" because the user got one digit (1) right and in the right position (a bull) and one digit (3) right but in the wrong position (a cow). The guessing continues until the user gets four bulls, that is, has the four digits correct and in the correct order.



!(The End)