SIENA*college*
Computer Science

Computer Science 120
Introduction to Programming
Siena College
Spring 2011

Topic Notes: Introduction and Overview

Welcome to Introduction to Programming!

Why take this course?

- *Computer programming* is the act of creating a *computer program*, which is a set of instructions that can be understood by a computer.
- Much of our time this semester will be spent developing your programming skills.

So why should you learn about computing and computer science, and in particular, programming?

- Programming skills may be applied in other areas.
- The experience of programming helps develop problem solving skills, in particular, the ability to deal with complexity.
- Computer technology is pervasive and computational thinking is pervading society. For example, biologists are beginning to model cells as distributed computing devices that communicate over membrane boundaries. Nearly every field of science now has a "computational" branch of that science which is becoming as important as the experimental and theoretical branches. Moreover, the impact of the computer on society is huge and will continue to accelerate over the years.
- Most programming is done by non-computer scientists. Customizing of tools is possible by those who understand programming. This can range from programming complicated queries on databases to running complex simulations with spreadsheets. The techniques you learn in any of these contexts can usually be applied to others.
- It's a fascinating challenge to teach the computer how to solve hard problems.

Consider how radically computing technology has changed the way we do things in just a little more than half a century.

- Communication: The Internet and world-wide web, fast and cheap communication worldwide.
- Productivity: word processing, spreadsheets, *etc.*.

- Creative arts: digital cameras, camcorders, digital image manipulation.
- Entertainment: CD, DVD, BluRay, MP3 technologies.
- Infrastructure: there are computer chips in your watch, toaster, car, phone, game system, iPod, *etc.*.

Computing is clearly is an exciting area!

Amazingly, computers today are fundamentally the same as computers of the 1950's. In the 1950's computers were primarily used as giant calculators, solving military and scientific problems. It was originally expected that there would be no need for more that just a few computers – especially given their cost and since computer programmers were expected to have a Ph.D. in mathematics or the sciences. (Computer time was expensive - people were relatively cheap.)

If you look closely enough at the internal circuitry, you will discover that there are basically only two things a computer can do:

1. perform numeric operations (addition, multiplication, comparison...)
2. follow a series of instructions that describes which operations to perform and when to perform them

The difference between today's computers and those built in the 50's is that today's are:

- much faster,
- much cheaper,
- much smaller,
- filled with much more memory, and
- they can communicate with other computers.

While the scale of these changes has been dramatic, the changes themselves are not fundamental.

Rather, what has happened is that the size, cost, and speed of computers has changed in such a way that we can now conceive of uses for computers that would have been ridiculously expensive in the early years or even just a few years ago. At least as important, we now know how to program computers in ways that are much simpler than those used in the early days.

It seems likely that part of what early pioneers failed to appreciate about computers was the power of numbers. The very name that its inventors gave the computer suggests that they thought the primary applications for computers would be processing numbers as numbers for the purpose of science and engineering. They did not recognize or appreciate the fact that numbers can be used to encode just about any information one might want to process.

- There are many examples where numbers are used to represent information but the values of the numbers used have little significance.
  - ZIP/postal codes,
  - social security or other identifying numbers,
  - license numbers, ...
- By associating a numeric value with each letter of the alphabet (a=1, b=2, etc.) we can construct a numerical encoding of any textual information.
- When a computer manipulates numbers of this sort, it is really being used as a symbol or information processor rather than a "computer".

It is the fact that just about any information can be encoded using numbers that gives the computer the ability to be a universal information processor. As a result, the encoding of information forms an important part of all of computer science. We refer to it as "data structures". We will consider some basic data structure here, and those who go on in CS will take CS 211, which focuses on data structures.

Recognizing the power of numbers as symbols isn't quite enough to explain the amazing impact computers have had.

- Simple, hand-held calculators can process numbers too, but they have not had the impact of computers.
- The computer's other, fundamental capability — the ability to follow pre-supplied instructions or programs — provides the rest of the explanation.
  - The ability to change programs makes computers flexible. They can be adapted to new tasks without being rebuilt.
  - The abilty to follow programs makes it possible to exploit the speed of a computer. If the instructions could not be pre-supplied, the computer would spend most of its time waiting for a slow human to press the next control button.

Algorithms

Our main focus in this course will be learning how to prepare the instructions needed to ensure that a computer can perform a particular task.

The instruction presented to a computer must be significantly different from the sorts of instructions we might prepare for another human being. Computers have no common sense or intuition. They can only perform a task if the instructions provided are accurate and totally unambiguous. It must be possible to follow the instructions without in any sense understanding their ultimate purpose

Such a set of instructions is called an *algorithm*.

- The instructions in an algorithm must specify what to do rather than what can or might be done. Thus, the "instructions" for a board game or a card game would not be considered an algorithm. The instructions found in a cook book recipe are much more similar to the instructions that must be included in an algorithm.

Many aspects of algorithms are studied in Computer Science:

- the design of languages for expressing algorithms,
- the mathematical analysis of the correctness of algorithms, ...

Our concern will simply be to learn how to write good algorithms (*i.e.*, how to write good instructions) and put them in a form (a program) that the computer can execute. To appreciate why it might take as long as a semester to accomplish this, consider:

- Humans, in general are very good at writing bad instructions.
  - Have you ever tried to assemble something complex by following the manufacturer's directions?
  - Have you ever tried to register for courses?
- Humans are, in general, very good at following bad instructions.
  - Think about a recipe that in the first step calls for preheating the oven and the second step calls for refrigerating a mixture of ingredients overnight. How many people would be silly enough to leave the oven on overnight?
- Computers are (always) very good at following instructions exactly.
  - A computer would definitely leave the oven on overnight.

Learning to write good algorithms is a skill. While we can provide advice on how to go about it, there are no set rules that will always enable you to produce a good algorithm for a given task. Experience is the best way to develop the needed skills.

At the same time that you are learning this skill, we will also be teaching you a language you will have to use to communicate your algorithms to a computer, Java.

- A good analogy of the process you are undertaking would be the process of learning a game like chess.
  - You have to first learn the legal moves for each chess piece. This is like learning the rules of the Java language. It isn't that hard but it isn't enough to make you a chess player (or a programmer).
  - Then, you have to learn strategy. You can read about other players approaches, but nothing beats practice.