What is Computer Science?

What is it that distinguishes it from the separate subjects with which it is related? What is the linking thread which gathers these disparate branches into a single discipline? My answer to these questions is simple --- *it is the art of programming a computer*. It is the art of designing efficient and elegant methods of getting a computer to solve problems, theoretical or practical, small or large, simple or complex.

C.A.R. (Tony) Hoare
Why is programming fun?

What delights may its practitioner expect as a reward?

First is the sheer joy of making things

Second is the pleasure of making things that are useful

Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts

Fourth is the joy of always learning

Finally, there is the delight of working in such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff.

Fred Brooks
Efficient Programming

- **Designing and building efficient programs efficiently requires knowledge and practice**
  - Hopefully the programming language helps, it’s not intended to get in the way
  - Object-oriented concepts, and more general programming concepts help in developing programs
  - Knowledge of data structures and algorithms helps

- **Tools of the engineer/scientist programmer**
  - A library or toolkit is essential, don’t reinvent the wheel
  - Someone must build the tools
  - Programming is not just art, not just science, not just engineering
Course Overview

● Lectures, Recitations, Quizzes, Programs
  ➢ Recitation based on questions given out in previous week
    • Discuss answers, answer new questions, small quiz?
    • More opportunities for questions to be answered.
  ➢ Lectures based on readings, questions, programs
    • Online quizzes used to motivate/ensure reading
    • In-class questions used to ensure understanding
  ➢ Programs
    • Theory and practice of data structures and OO programming
    • Fun, practical, tiring, ...

● Ask questions, give feedback, provide guidance
  ➢ Arno Penzias and others say "ask good questions"
Tradeoffs

- This course is about all kinds of tradeoffs: programming, structural, algorithmic
  - Programming: simple, elegant, quick to run/to program
    • Tension between simplicity and elegance?
  - Structural: how to structure data for efficiency
    • What issues in efficiency? Time, space, programmer-time
  - Algorithmic: similar to structural issues

- How do we decide which choice to make, what tradeoffs are important?
See readwords.cpp

- This reads words, how can we count different/unique words?

```cpp
tvector<string> list;
string filename, word;
cin >> filename;
ifstream input(filename.c_str());
CTimer timer;
timer.Start();
while (input >> word) {
    list.push_back(word);
}
timer.Stop();
cout << "read " << list.size() << " words in ";
cout << timer.ElapsedTime() << " seconds" << endl;
```
Tracking different/unique words

- We want to know how many times ‘the’ occurs
  - Do search engines do this? Does the number of occurrences of “basketball” on a page raise the priority of a webpage in some search engines?
    - Downside of this approach for search engines?

- Constraints on solving this problem
  - We must read every word in the file (or web page)
  - We must search to see if the word has been read before
  - We must process the word (bump a count, store the word)

- Are there fundamental limits on any of these operations? *Where should we look for data structure and algorithmic improvements?*
Search: measuring performance

- **How fast is fast enough?**

```cpp
bool search(const tvector<string> & a,    
    const string & key)
// pre:  a contains a.size() entries
// post: return true if and only if key found in a
{
    int k; int len = a.size();
    for(k=0; k < len; k++)
        if (a[k] == key) return true;
    return false;
}
```

- **C++ details: parameters? Return values? Vectors?**
- **How do we measure performance of code? Of algorithm?**
  - Does processor make a difference? PIII, G4, ???
Tradeoffs in reading and counting

- If we read all the words, then sort them, can we determine unique words?
  - frog, frog, frog, rat, tiger, tiger, tiger, tiger

- If we look up words as we're reading them and bump a counter if we find the word, is this slower than previous idea?
  - How do we look up word, how do we add word

- Are there kinds of data that make one approach preferable?
  - What is best case, worst case, average case?
Structuring data: readwords4.cpp

- Search for a word using binary search
  - Differences from sequential/linear search?
  - What’s a precondition for binary search to work?

- How can we store new words so that binary search will work?
  - Add to end of vector and sort the vector
  - Add to end of vector and shift (down) until location found
  - Advantages of one method over another?

- What about the C++ details in using a struct/class to store data, how are comparisons made?
Flexibility

- About Linux compared to Windows
  - “All this flexibility comes with a price: complexity. Many vendors are realizing that ease of use is paramount for most and have developed standard "distributions" consisting of Linux bundled with a wide range of necessary software, from word processors and spreadsheets to personal organizers. Some of these distributions are: RedHat (perhaps the best known), SuSE, Debian, Slackware and Mandrake.”

  http://imprint.uwaterloo.ca/issues/022500/3Science/science05.shtml
Flexibility

- About FPGAs compared to general computers
  - “The most common type of computing system is the general-purpose processor. Under this model, the hardware of the system is limited to merely a few basic tasks. By combining and building off of these operations, a general-purpose computer can perform a much larger number of operations than it was originally designed to handle. Which is why the general-purpose computer is so flexible. However this flexibility comes with a price. For any specific application, the general-purpose processor will perform poorly when compared to a custom hardware implementation”

http://www.tomfry.com/thesis.doc
Toward understanding inheritance

- **Consider Yahtzee program**
  - Several kinds of score-card entries: aboveline, fullhouse,...
  - What do we do to add a new kind, or change definition
    - Edit working program, add code to code that already works
    - Not haphazard, but dangerous: if it ain’t broke, don’t touch

- **The “open-closed” principle of program building/design**
  - Programs/classes should be open for extension, but closed to modification (Bertrand Meyer and Robert Martin)
  - How do we change code without altering it?
    - This is a goal, in practice we modify some, but minimize
- **Inheritance lets us realize the principle (in theory)**
Use interfaces, not implementations

- See yahtzee program, ScoreEntry, DiceGroup
  - Subclasses: AboveLine, FullHouse, ThreeKind, ...
  - Subclasses: CupOfDice and FixedDice (game and testing)

- We have parent, base, or super-class for interface
  - Child or Subclasses extend the parent class (implement interface)
  - pointers to parent class, which can point to child object
    - I’m pointing at an animal (could be marsupial or bird)

- Child/sub classes don’t have access to private data/functions, but the data are there (alternative: use protected)
  - See ScoreEntry for details
Syntactic details

- **Functions in parent class that are changeable in subclasses must be declared as `virtual`**
  - The “real” function is determined at runtime
  - In parent/base/super class make all functions virtual
  - We must have a virtual destructor that’s implemented

- **For interface only functions, that have no default version**
  - Pure virtual functions, aka abstract functions
  - Syntax requires `= 0` in `.h` file
  - Can’t instantiate an object of an abstract (base) class
    - Doesn’t matter, really, we use pointers to objects