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
"...I first learned about recursive procedures and saw how to program the sorting method which I had earlier found such difficulty in explaining...Due credit must be paid to the genius of the designers of ALGOL 60 who included recursion in their language and enabled me to describe my invention so elegantly to the world."

"I have learned more from my failures than can ever be revealed in the cold print of a scientific article and now I would like you to learn from them, too....failures are much more fun to hear about afterwards; they are not so funny at the time."
Efficient *design, programs, code*

Using the language: C++ (or Java, or Python, or ...), its idioms, its idiosyncracies

Object-oriented design and patterns. Software design principles transcend language, but ...

Know data structures and algorithms. Trees, hashing, binary search, sorting, priority queues, greedy methods, ...

Engineer, scientist: what toolkits do you bring to programming? Mathematics, design patterns, libraries --- standard and Tapestry
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, ...
    - Weekly programs and longer programs
- **Exams/Tests**
  - Semester: closed book
  - Final: open book
Questions

If you gotta ask, you’ll never know
Louis Armstrong: “What’s Jazz?”

If you gotta ask, you ain’t got it
Fats Waller: “What’s rhythm?”

What questions did you ask today?
Arno Penzias
Tradeoffs

Programming, design, algorithmic, data-structural

Simple, elegant, quick, efficient: what are our goals in programming? What does XP say about simplicity? Einstein?

Fast programs, small programs, run anywhere-at-all programs. Runtime, space, your time, CPU time...

How do we decide what tradeoffs are important? Tension between generality, simplicity, elegance, ...
Code/hierarchy in readwords.cpp

- Count number of different words in a vector, how?

```cpp
class WordStats
{
    public:
        WordStats(){}  // no state, nothing to do
        virtual ~WordStats(){}  // make compiler happy
        virtual int uniqueCount(tvector<string> v);
};

class SetStats : public WordStats
{
    public:
        int uniqueCount(tvector<string> v) {...}
};
```
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)
  - Search for the word? Avoid counting twice? Store?
  - Are there fundamental limits on any of these operations?

*Where should we look for data structure and algorithmic improvements?*
What does this do? Why is it wrong?

```cpp
int WordStats::uniqueCount(tvector<string> v) {
    // post: returns # unique/different words in v
    int count = 0;
    for (int k = 0; k < v.size(); k++) {
        string word = v[k];
        count++;
        for (int j = k + 1; j < v.size(); j++) {
            if (v[j] == word) {
                string temp = v[j];
                v[j] = v[v.size() - 1];
                v[v.size() - 1] = temp;
                v.pop_back();
            }
        }
    }
    return count;
}
```
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? G5? Itanium? 64-bit?
Tradeoffs in processing and counting

- Read words, then sort, determine # unique words?
  - frog, frog, frog, rat, tiger, tiger, tiger, tiger

- If we look up words one-at-a-time and bump counter if we haven't seen a 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?
Benefits of inheritance, interfaces

- Consider adding new algorithm for determining unique word count

```cpp
void doStats(tvector<string> v, WordStats& stats)
{
    CTimer timer;
    timer.Start();
    int unique = stats.uniqueCount(v);
    timer.Stop();

    cout << unique << " different words in ";
    cout << timer.ElapsedTime() << " sec." << endl;
}
```

- Why can we pass different kinds of objects to doStats?
  - Why is this an advantage?
  - Inheritance and late/dynamic binding
Why inheritance?

- Add new shapes easily without changing much code
  - Shape * s1 = new Circle();
  - Shape * s2 = new Square();
- abstract base class:
  - interface or abstraction
  - pure virtual function
- concrete subclass
  - implementation
  - provide a version of all pure functions
- “is-a” view of inheritance
  - Substitutable for, usable in all cases as-a

User’s eye view: think and program with abstractions, realize different, but conforming implementations,

don’t commit to something concrete until as late as possible
Example of inheritance

- What is behavior of a shape?

```cpp
void doShape(Shape * s) {
    cout << s->area() << endl;
    cout << s->perimeter() << endl;
    s->expand(2.0);
    cout << s->area() << endl;
    cout << s->perimeter() << endl;
}

Shape * s1 = new Circle(2);
Shape * s2 = new Square(4);
Shape * s3 = new Rectangle(2,5);
doShape(s1); doShape(s2); doShape(s3);
```
Inheritance (language independent)

- **First view: exploit common interfaces in programming**
  - Streams in C++, iterators in Tapestry classes
    - Iterators in STL/C++ share interface by convention/templates
    - Implementation varies while interface stays the same

- **Second view: share code, factor code into parent class**
  - Code in parent class shared by subclasses
  - Subclasses can *override* inherited method
    - Subclasses can override and call

- **Polymorphism/late(runtime) binding (compare: static)**
  - Function actually called determined when program runs, not when program is compiled
Who is Alan Perlis?

- It is easier to write an incorrect program than to understand a correct one
- Simplicity does not precede complexity, but follows it
- If you have a procedure with ten parameters you probably missed some
- If a listener nods his head when you're explaining your program, wake him up
- Programming is an unnatural act
- Won first Turing award

http://www.cs.yale.edu/homes/perlis-alan/quotes.html