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

Review/Preview: Anagrams/Jumbles

- Brute-force approach to finding anagrams/solving Jumbles
  - Brute-force often thought of as “lack of thought”
  - What if the better way requires too much thought?
  - What if there’s nothing better?
  - nelir, nelri, neilr, nei, nel, nerli, neril, nlerin, nler, nreil, nleri, nri, nirel, nielr, nieri, nirol, niel, nleri, nirel, niel, nirol, nleri, nirel
  - What’s the problem here?
  - Is there a better method?

Brute force? permana.cpp

```cpp
// find anagram of word in wordSource
// list is a vector [0, 1, 2, ..., n]
Permuter p(list);
int count = 0;
string copy(word); // makes copy the right length

{
  p.Current(list);
  for(k=0; k < list.size(); k++)
  {
    copy[k] = word[list[k]];
  }
  if (wordSource.contains(copy))
  {
    cout << "anagram of " << copy << endl;
    break; // find first anagram only
  }
}
```

Quantifying brute force for anagrams

- On one machine make/test a word takes $10^{-5}$ seconds/word
  - 9! is 362,880, how long does this take?
  - What about a ten-letter word?

- We’re willing to do some pre-processing to make the time to find anagrams quicker
  - Often find that some initialization/up-front time or cost saves in the long run
  - We need a better method than trying all possible permutations
  - What properties do words share that are anagrams?
Preliminaries: C++ in permana.cpp

- What is a dictionary?
  > What is a class, what are the methods, why use it?
  > What properties of the class do methods depend on, class invariants?

- What is a tvector and why is it used instead of an array?
  > How are elements added to the vector?
  > Differences between tvector and vector (STL class)?

- What is a Permuter and how does it work?
  > Where is information about this class found?
  > What patterns of use does a permuter exhibit?

Toward a faster anagram finder

- Words that are anagrams have the same letters; use a letter fingerprint or signature to help find anagrams
  > Count how many times each letter occurs:
    - “teacher” 1 0 1 0 2 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
    - “cheater” 1 0 1 0 2 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0

- Store words, but use fingerprint for comparison when searching for an anagram
  > How to compare fingerprints using operator ==
  > How to compare fingerprints using operator <

- How do we make client programmers unaware of fingerprints? Should we do this?

OO and C++ features we’ll use

- We’ll use an adapter or wrapper class called Anaword instead of a string
  > Clients can treat Anaword objects like strings, but the objects are better suited for finding anagrams than strings
  > The Anaword for “bear” prints as “bear” but compares to other Anaword objects as 11001000000000000100000000

- C++ allows us to overload operators to help, not necessary but good cosmetically
  > Relational operators == and <
    > What about other operators: >, <=, >=, and !=
  > Stream operator <<

- How do we implement overloaded operators?

More word games: boggle

- If letter connect horizontally, vertically, diagonally, and cannot be re-used, find words on board:
  
  SYER | ant | (3,2)(3,3)(2,3)
  
  EUOZ | exam|
  
  EXMT | rotate|
  
  UTAN | tease|

- How do we write a program to do this?
  > What are some of the issues?
  > What do we need to know about program requirements?
Boggle choices

- There’s a board and a dictionary
  - Look up “words” from the board in the dictionary
  - Look up words from the dictionary on the board
- What operations must the dictionary support if we look up “words” from the board? Vice-versa?
  - The dictionary, viewed through its operations rather than implementation is an Abstract Data Type (ADT)
    - Language independent concept realized in C++ by classes
  - How can we structure the dictionary to support these operations efficiently?

BoggleFinder, find word on a board

- In some languages a seemingly naïve, quasi-brute force algorithm may be good enough
  - In C or C++, looking up every word in a dictionary on the board is fast enough for HCI
  - In Java this same method wasn’t fast enough in 1996, different today?
- Given a word, where do you start looking it up on the board?
  - Try all 25 locations on a 5x5 board? Any obvious, easy to implement improvements?
- After finding the first letter, look for next letter, but where?
  - Look at adjacent locations of first letter
  - Look at all second letter locations, see which are adjacent
  - Are there differences?

Backtracking to solve problems

- Backtracking is used when certain criteria are met:
  - Sequence of steps leads to a solution, steps are similar
    - In Boggle sequence is find next letter in an adjacent location
  - For each step there is at least one, usually more, ways of taking the step
    - In Boggle the next letter could be found at several locations, (or the next adjacent location could be the letter)
  - We can tell when the steps are finished
    - In Boggle when all letters of a word are found
- Algorithm: take a tentative step, but make sure the step is undoable. If the step leads to a solution, done, otherwise undo the tentative step and try the next step alternative

Pointers

- Pointers are essential in many programming applications
  - Indirect references are often useful
    - Publish your email as foo@hotmail.com, but forward it to wherever you “really” are as you change jobs, for example
  - Allow data to be shared rather than duplicated
    - Sort a list of people/grades by name and by grade, we can maintain one list of people, and two lists of indexes, one sorted by name, one sorted by grade
  - Facilitate inheritance
    - Essential for OO
  - Implement data structures
    - Lists, trees, ...
Syntax and semantics of pointers

```c
void foo()
{
    string s("hello");
    string * sp = new string("world");
    string * sp2;
    int slen = s.length();
    int splen = sp->length();
    // splen = (*sp).length();
    int splen2 = sp2->length();
}
```

- Memory allocated dynamically using `new`
  - What happens to `s` when `foo` terminates?
- Dereference a pointer using `*`, get at the object pointed to
  - `a->` is shorthand for `(*a)`
- Pointers that don’t point at something are BIG TROUBLE

Vectors of pointers

```c
void readWords(istream& input, tvector<string>& list)
// post: all words in input stored in list
{
    string word;
    while (input >> word)
    {
        list.push_back(word);
    }
    cout << "read " << list.size() << " words" << endl;
}
```

- What changes if we use `tvector<string *>` instead?
- What happens if we use code below which uses the address-of operator for vector of pointers (don’t do this at home)
  ```c
  list.push_back(&word);
  ```

Sidebar: implementing swap in C

- Unlike C++, there are no reference parameters in C
  - Simulate pass-by-reference using pointers, what happens to actual parameters `x` and `y` in code below?

```c
void swap(string& a, string& b)
{
    string temp = a;
    a = b;
    b = temp;
}
```

```c
int main()
{
    string x("first"), y("second");
    swap(x, y);
}
```

Implementing swap in C

- In C we must pass pointers, and use address-of operator to simulate reference parameters, `is the picture different`?
  - In C++ the pointers are hidden, harder to make mistakes?

```c
void swap(string * a, string * b)
{
    string temp = *a;
    *a = *b;
    *b = temp;
}
```

```c
int main()
{
    string x("first"), y("second");
    swap(&x, &y);
}
```