CS 180 - C++ tidbits

**Announcements**
- Lab tomorrow
- No class Monday
- Checkpoint due Tuesday
  - Read Ch. 1.6 of text
Review: Types of Variables

1. Value - standard

2. Reference - creates a variable that references another variable

3. Pointer - null value, easy to change what it is pointing at, delete
Sample Code: What is output?

```cpp
char ch = 'Q'; // Creates value variable
char* p = &ch; // creates a pointer
cout << *p << ch;
ch = 'Z';
cout << *p << ch;
char** s;
s = *p;
*(s++) = ' Z';  // s will contain for Z

Output: Q Z
```
Caution: Common Error

```c
int i, j;
int j = 36;
```

meant \( i = \&j \).

What is the error?

didn't point a pointer variable to memory address

(more examples in text & trans. guide)
Structures: (Legacy from C)

useful for holding collections of objects

Ex:

```c
enum MealType {NO_PREF, REGULAR, VEG};

struct Passenger {
    string name;
    MealType mealPref;
    bool isFreqFlyer;
    string FreqFlyerNo;
};
```
Using Structures

Structures can then be used inside the program:

```
Passenger pass = {'John Smith', VEG, true, '1234'};
pass.mealPref = REGULAR;
```
Another example:

```java
Passenger *p;  // tells compiler to create passenger
p = new Passenger;
p -> name = "Barbara Wright";
p -> mealPref = NO_PREF;
p -> isFreqFlyer = false;
p -> freqFlyerNo = "NONE";
(*p). freqFlyerNo = "";  // - -
```
"Larger" Projects: Our Credit Card Class

Code provided for:
- CreditCard.h
- CreditCard.cpp
- TestCard.cpp

See p. 49 of text, or class website

Note: Makefile → type "make" create Test Card
Ch 3 - How to analyze running time?

So how?

- less memory
- usability
- fast
- reliability

Don't want to just time things.

Downsides:
- computers have different architectures, OS, programming languages.
Counting primitive operations

Identify high-level primitive operations independent of language compilers, OS, or computer.

Ex:
- comparisons
- create variables, store value
- addition
- multiplication
- branching
Ex: (pseudocode to find max in an array)

A way to describe an algorithm that is language-independent.

Algorithm arrayMax(A, n):

  Input: An array $A$ of $n \geq 1$ numbers
  Output: The maximum element of $A$

  $\text{currentMax} \leftarrow A[0]$
  for $i < 1$ to $n-1$
    if $\text{currentMax} \leq A[i]$, then
      $\text{currentMax} \leftarrow A[i]$
  return $\text{currentMax}$
Advantage of pseudocode:
- independent of language
- easy to read and translate to any language

Ex: (in C++)

```cpp
int arrayMax(int A[], int n) {
    int currentMax = A[0];
    for (int i = 1; i < n; i++) {
        if (currentMax < A[i]) {
            currentMax = A[i];
        }
    }
    return currentMax;
}
```
Counting operations:

Algorithm arrayMax(A, n):
  Input: An array A of n ≥ 1 numbers
  Output: The maximum element of A

1. currentMax ← A[0] ← 1 operation
2. for i ← 1 to n-1 ← n-1 variable assignments +
3.   if currentMax < A[i] then ← 1 comparison
4.     currentMax ← A[i] ← 1 variable assignment
5. return currentMax ← 1 memory access

(best case)

Sum: min: 1 + 2(n-1) + n-1 + 1 = 3n-1
worst case: (3n-1) + n-1 = 4n-2
So how many operations in best (or worst) case?

- best: \( 3n - 1 \)
- worst: \( 4n - 2 \)
Average case versus worst case

\[ q \sim 3.5n \quad 4n - 2 \]

We use worst case

Why?

- hard to analyze average
- really want worst case