CS 180 - C++ tidbits

Note Title 9/2/2010

Announcements

- Lab tommorrow
- 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

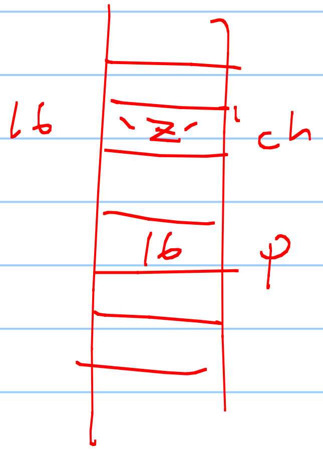
easy to change what it is pointing at

↑ delete

Sample Code : What is output?

```
char ch = 'Q';     //creates value variable
char* p = &ch;     //creates a pointer
cout << *p;
ch = 'Z';
cout << *p;
char * s;
s = *p;
 &(*p)
```

s will contain the ASCII # for Z



Output: QZ

Caution: Common Error

```
int* i;
int j(36);
i = j;
```

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:

```
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:

```
Passenger *p;
```

tells compler to create passenger + point p to it.

```
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

header file, contains private vars + lists the functions

see p.49 of text, or class website

Note: Makefile → type "make" create TestCard

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, compiler, OS, or computer

Ex:

- comparisons
- create variables, store value
- addition
- multiplication
- branching

Ex: (pseudocode to find max in an array)

↑ 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
                              assignment operator
  currentMax ← A[0]
  for i ← 1 to n-1
        if currentMax < A[i] then
              currentMax ← A[i]
  return currentMax
```

Advantage of pseudocode:

- independent of language
- easy to read & translate to <u>any</u> language

<u>Ex:</u> (in C++)

```
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

currentMax ← A[0]                ← 1 operation
for i ← 1 to n-1                 ← n-1 variable assignments & n-1 comparisons
    if currentMax < A[i] then    ← 1 comparison
        currentMax ← A[i]        ← 1 variable assignment
return currentMax                ← 1 memory access
```

Sum: (best case) 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?

$$\begin{array}{r} \text{best} \\ \text{worst} \end{array} \begin{bmatrix} 3n-1 \\ 4n-2 \end{bmatrix}$$

~~Average case~~ versus worst case

$\sim 3.5n$ $4n-2$

We use worst case

why?

- hard to analyze averge
- really want worst case