CS 161
Intro to CS I

Addresses, Pointers, and References
Odds and Ends

• Continue to work on Assignment #4
• Exercise #5 due tonight

```
if stack.empty():
    string = input("string ( ) ")
    string = string.split()
    return True
else:
    return False
```

Bad!!
Pointer and References Cheat Sheet

- *
  - If used in a declaration (which includes function parameters), it creates the pointer.
    - Ex. int *p;  //p will hold an address to where an int is stored
  - If used outside a declaration, it dereferences the pointer
    - Ex. *p = 3;  //goes to the address stored in p and stores a value
    - Ex. cout << *p;  //goes to the address stored in p and fetches the value

- &
  - If used in a declaration (which includes function parameters), it creates and initializes the reference.
    - Ex. void fun(int &p);  //p will refer to an argument that is an int by implicitly using *p (dereference) for p
    - Ex. int &p=a;  //p will refer to an int, a, by implicitly using *p for p
  - If used outside a declaration, it means “address of”
    - Ex. p=&a;  //fetches the address of a (only used as rvalue!!!) and store the address in p.
C++ Reference vs. Pointer

```cpp
#include <iostream>
#include <string> // C++ strings
using namespace std;

// Construct local string object to return to main, but expensive
string get_string()
{
    string s;
    cout << "Enter a string to return: ";
    cin >> s;
    return s;
}

// Only works in C++, pass by reference
void get_string(string &str)
{
    cout << "Enter a string for our ref: ";
    cin >> str; // implicitly * (or dereference) to go to s in main
}

// Works in C/C++
void get_string(string *str)
{
    cout << "Enter a string for our pointer: ";
    cin >> *str; // explicit dereference or take me there
}
```

"strings2.cpp" 38L, 804C written
C++ Reference vs. Pointer

```cpp
// works in C/C++
void get_string(string *str){
    cout << "Enter a string for our pointer: ";
    cin >> *str; // explicit dereference or take me there
}

int main() {
    string s;

    // we can call three different functions with the same name
    // argument types determine which function is called, overloading
    s = get_string(); // capture returned string
    cout << s << endl;

    get_string(s);  // implicitly send address
    cout << s << endl;

    get_string(&s); // explicitly pass the address
    cout << s << endl;

    return 0;
}
```
In-class Exercise #3

Understanding Pointers

• Create a pointer to a double, i.e. `double *d;` and three doubles `d1`, `d2`, and `d3` that get the values `7.8`, `10.0`, and `.009`.

• Now, set the pointer, `d`, to point to each double variable, `d1`, `d2`, and `d3`, printing the address and contents of each double variable along the way.
```cpp
#include <iostream>
#include <cstdlib> // gives you NULL pointer

using namespace std;

int main(){
    double *d=NULL, d1=.78, d2=1.2;
    // how do you make a pointer point to something
    // common mistakes:
    //   *d=d1 (core dump because you try to go to bad address NULL)
    //   *d=&d1 (compiler error because trying to put double * in double)
    //   d=d1 (compiler error because trying to put double in double *)
    d=&d1; // always fill contents before you go somewhere

    cout << &d << endl; // where d lives
    cout << &d1 << endl; // where d1 lives
    cout << d << endl; // contents of d is d1's address
    cout << *d << endl; // take me to the address in my contents, dereference
    cout << d1 << endl; // print d1's contents

    return 0;
}
```

In-class Exercise #3

Understanding Pointers

• What if you made a pointer that points to a pointer to a double, i.e. `double **dp`? Now, set dp to point to d, and use dp to print the address and contents of each double variable!!!
Monday Demo...