CS 161
Intro to CS I

More Functions and
Pass by Value vs. Reference
Odds and Ends

• Assignment #4 design due Sunday, 11:59pm
  – Submit into Assignment on Canvas!!!
• Demo Assignment 3
• Pick up Exam 1 in KEC 1174
C/C++ Pointers

void swap(int *, int *);
int main() {
    int a=5, b=10;
    swap(&a, &b);
    cout << "a: " << a << " b: " << b;
}

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
C++ Pass by Reference

```c++
void swap(int &, int &);

int main() {
    int a=5, b=10;
    swap(a, b);
    cout << "a: " << a << "b: " << b;
}

void swap(int &x, int &y) {
    int temp = x;
    x = y;
    y = temp;
}
```

---

Diagram:
- `x` is reference to an int
- Can't access `x`'s memory
- Implicit dereference
- `Addr1` points to `a`
- `Addr2` points to `b`
- `Addr3` points to `&x`
- `Addr4` points to `&y`

---

Oregon State University
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.
#include <iostream>

using namespace std;

int main() {
    int s=20, t=10;
    //make a c++ reference
    int &r=t; //make it refer to something at time of creation
    //make a c/c++ pointer
    int *p;

    p=&s; //p points to s, store address of s into p
    cout << "s contents: " << *p << endl; //dereference, go to contents of s
    *p=30; //deref, go to s and store 30
    cout << "s contents: " << s << endl; //look s changed using p

    cout << "p address: " << &p << endl; //p lives somewhere
    cout << "s address: " << &s << " or " << p << endl;

    cout << "t contents: " << r << endl; //r is really t now
    cout << "r address: " << &r << endl; //r is really t, so address of t
    cout << "t address: " << &t << endl; //matches address of t

    //change who p points to
    p=&t; //p points to t now!
    cout << "t contents: " << *p << endl; //deref p, which is t
    cout << "p contents: " << p << endl; //p contains t's address

    r=s; //can't change who r refers to because constant, changes t's value
    cout << "t contents: " << t << endl;
    //r=&s; //can't put int * into an int
    //&r=&s; //illegal lvalue because you can't change an address

    return 0;
}
Group Exercise...

Write two functions (one using references and one using pointers) that gets two C++ strings of characters as input from the user.
```
#include <iostream>
#include <string>
using namespace std;

void fun(string *s, string *t) {
    cin >> *s;
    cin >> *t;
}
void fun(string &s, string &t) {
    cin >> s;
    cin >> t;
}

int main() {
    string str, str1;
    fun(&str, &str1);
    cout << str << " " << str1;
    fun(str, str1);
    cout << str << " " << str1;
    return 0;
}
```