CSE 220 – C Programming
Fall 2015

Pointers (...)

Notes

- Good news: No more Homework

- Remaining Labs: Working on Project

- Next Monday: Finishing Materials
  
  Next Wednesday: Solving sample questions and working on project.

- Extra office hours for project

- Extra points for those who did not do well on midterm exam.
Pointers as Arguments
Passing Arguments: Call by Value

```cpp
int main()
{
    ... 
    int x = 1, y = 10;
    int z = sum(x, y);
    ...
    return 0;
}

int sum(int a, int b)
{
    int s = a + b;
    return s;
}
```
int main()
{
  ...
  int x = 1, y = 10;
  int z = sum(x, y);
  ...
  return 0;
}

int sum(int a, int b)
{
  int s = a + b;
  return s;
}
int main()
{
    ... 
    int x = 1, y = 10;
    int z = sum(x, y);
    ...
    return 0;
}

int sum(int a, int b)
{
    int s = a + b;
    return s;
}
Passing Arguments: Call by Value

- Copy of actual argument passed
- Considered "local variable" inside function
- If modified, only "local copy" changes
  Function has no access to "actual argument" from caller
- This is the default method
C passes arguments by value

What if we want the function to return multiple values (not global variables :-))? 

What if the data to be passed is too large to be copied (efficiency + memory utilization)?

Solution: pass a pointer to the value (call be reference)
Pointers as Arguments

After calling the function `swap`, the values of `x` and `y` remain the same

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

int x = 10, y = 20;
swap(x, y);
// x and y are same as before
```
Pointers as Arguments

Instead of passing the value, pass a pointer or reference to the actual parameter.
Pointers as Arguments

Pass a *pointer* or *reference* to the actual parameter

```c
void swap(int *ap, int *bp) {
    int temp = *ap;
    *ap = *bp;
    *bp = temp;
}
```

```c
int x = 10, y = 20;
...
swap(&x, &y);
```

/* swap expects a pointer: so pass &x instead of x, &y instead of y */
Pointers as Arguments

/* swap expects a pointer: so pass &x instead of x, &y instead of y */

void swap(int *ap, int *bp) {
    int temp = *ap;
    *ap = *bp;
    *bp = temp;
}

int x = 10, y = 20;
swap(&x, &y);
void swap(int *ap, int *bp) {
    int temp = *ap;
    *ap = *bp;
    *bp = temp;
}

int x = 10, y = 20;
swap(&x, &y);

/*Note: NO direct access to variables x and y inside swap (local scope) but indirect access using their addresses*/