CS 161
Intro to CS I
Pointers and Static vs. Dynamic Memory
Variables vs. Pointers

• Value Semantics
  – Values stored directly
  – Copy of value is passed
  int i, j=2;
  i=j;

• Pointer Semantics
  – Address to variable is stored
  – Copy of address is passed
  int *i, j=2;
  i=&j;
References vs. Pointers

```cpp
#include <iostream>
#include <cstdlib>

using namespace std;

int main() {
    int a=3, d=5;
    //int *b=&a; //this is the setup between args and params
    int &c=a; //make c refer to a from here on out...
    int *b=NULL;
    b=&a;

    cout << &a << " " << a << endl;
    if(b!=NULL) //check to make sure you can dereference b, i.e. *b
        cout << &b << " " << b << " " << *b << endl;
    cout << &c << " " << c << endl;
    c=d; //changes contents of a, not make c refer to d
    b=&d;

    cout << &a << " " << a << endl;
    cout << &b << " " << b << " " << *b << endl;
    cout << &c << " " << c << endl;

    return 0;
}
```
What if we don’t have the j?

• We need to create the address space.
• How do we do this?
  – `new` type;
• For example:
  ```
  int *i = NULL;
  i = new int; //new returns an address
  *i = 10;
  ```
• [http://cslibrary.stanford.edu/104/](http://cslibrary.stanford.edu/104/)
Static vs. Dynamic

• Static Semantics
  – Assign address of variable
    int *i, j=2;
    i=&j;

• Dynamic Semantics
  – Create memory
  – Assign memory to pointer
    int *i=NULL;
    i=new int;
    *i=2;
```cpp
#include <iostream>
#include <cstdlib>
using namespace std;
int * fun() {
    int *a=new int; //create memory on heap
    *a=20;
    //delete a; //don't do this because you free memory you are returning
    //DO NOT ever return memory on the stack from in a function!!!
    /*int a;
    cout << "a in fun is: " << a << endl;
    a=20;
    return &a;*/
    return a; //return memory on heap from within function
}
int main() {
    int a=3;
    int *b=NULL; //initialize to NULL when b doesn't point somewhere
    b=&a;
    //delete b; //cannot delete something off stack!
    b=fun();
    cout << b << " " << *b << endl;
    *b=10;
    cout << b << " " << *b << endl;
    //if we don't delete the heap memory before pointing to new memory
    //on the heap, we create a memory leak!!!
    delete b;
    b=fun();
    cout << b << " " << *b << endl;
    return 0;
}
```
Creating Memory in Functions

Advantages to Dynamic Memory

```cpp
int *i=NULL; //created in main function

i = create_mem(); //call in main

int * create_mem() {
    return new int;
}

OR

create_mem(&i); //call in main

void create_mem(int **m) {
    *m = new int;
}
```
What About Memory Leaks?

- What happens here...

```cpp
int main () {
    int *i=NULL; //created in main function
    while(1) {
        i = create_mem(); //call in main
    }
}

int * create_mem() {
    return new int;
}
```
Fixing Memory Leaks...

- What happens here...

... 

```cpp
int main () {
    int *i=NULL; //created in main function
    while(1) {
        i = create_mem(); //call in main
        delete i; //free memory that i points to, preventing mem leaks
    }
}

int* create_mem(){
    return new int;
}
```