CS 161
Intro to CS I

Pointers and Static vs. Dynamic Memory
Iterative Factorial

factorial(0) = 1;
factorial(n) = n\times(n-1)\times(n-2)\times\ldots\times(n-(n-1))\times 1;

long factorial(int n) {
    long fact;
    if(n==0)
        fact=1;
    else
        for(fact=n; n > 1; n--)
            fact=fact*(n-1);
    return fact;
}
Recursive Factorial

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

long factorial(int n) {
    if (n == 0)   // Base case
        return 1;
    else
        return n * factorial(n - 1);   // Recursive call
}
In-class Exercise

• Get into groups of 4 – 5.
• Write your own recursive int pwr() function that takes two integers as arguments and returns the integer result.
  – What does the function prototype look like?
  – Now, write the function definition...
Recursion Demo...
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.
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`, and print the address of each variable, along with the contents of the double variable using `d`. 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!!!
Reference/Pointer Demo...
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;
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/)
Stack vs. Heap

• Static vs. Dynamic
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;
    ```
Creating Memory in Functions

Advantages to Dynamic Memory
int *i=NULL; //created in main function

create_mem(&i); //call in main void
create_mem(int **m) {
    *m = new int;
}
OR
i = create_mem(); //call in main
int * create_mem() {
    return new int;
}
What About Memory Leaks?

• What happens here...

... 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...

...  
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;
}