# C++ Object model

- From last time:
  - pass by reference
  - array parameter passing
- **`freq.cpp`** example:
  - function prototypes
  - file organization (single file program)
  - practice with parameter passing
- Object model in C++
- Parameter passing for objects
- Defining classes

- All files in blue are in
  **`~csci455/code/04-11`**

# Announcements

- PA4 due on Wednesday
- lab 12 on Linked lists – Thur. lecture topic
- Check that your pa3 score was entered correctly.

# Review: Parameter passing

Can think of three kinds of parameters:

- IN
- OUT
- IN-OUT

- use pass by value for the first
- can use pass by reference for the second two
- if just one OUT param, can use return value of function

# Call by reference

- Call by reference for IN-OUT mode

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

int main() {
  int x = 10;
  int y = 20;
  swap(x, y);
  cout << x << " " << y << endl;
  return 0;
}
```

# Review: Call by reference vs. call by value

```
void foo(int & a, int b) {
   a = 100;
   b = 50;
}
int main() {
  int x = 10;
  int y = 20;
  foo(x, y);
  cout << x << " " << y << endl;
  return 0;
}
```

# Passing arrays as parameters

- Array is not copied, just like in Java:

```
void foo(int myarr[], int size) {
  for (int i = 0; i < size; i++) {
     myarr[i] = 50;
  }
}

int main() {
  int arr[20];
  foo(arr, 20);
  cout << arr[0] << endl;
  return 0;
}
```

# OUT parameters

- For OUT example do histogram example:

`freq.cpp`

- also to discuss file organization

# C++ object model

- Two ways to define objects in C++:

1. automatic ("on the stack")  [ default ]

1. dynamic (create with new)
   [ uses pointer syntax ]

# C++ object model

- Object that is a local variable:

```
void myFunc() {
  vector<int> v;
  int i = 17;
  v.push_back(3);
  v.push_back(17);
  v.push_back(5);
  v = vector<int>();  // re-inits
}
```

# Passing an object as a parameter

- Pass an object by value: the whole object gets copied:

```
// ex from lab
void printVals(vector<int> v) {

  for (int i = 0; i < v.size(); i++) {
    cout << v[i] << " ";
  }

}
```

- How can we avoid making a copy here?

# Pass by const-ref

- Can tell the compiler / client that the function doesn't change the object.
- But still get the efficiency of call-by-ref:

```
void printVals(const vector<int> & v) {

  for (int i = 0; i < v.size(); i++) {
    cout << v[i] << " ";
  }

}
```

- Use instead of call-by-value for objects
- for primitive types use pass by value

# Returning objects by value

- Same semantics for return objects by value:

```
// ex from lab
vector<int> readVals() {
  // reads data from user into a vector
}
. . .
vector<int> v;
v = readVals();
```

- The whole vector is copied back to caller
- How to do this without copying the whole vector?

# Objects as IN OUT params

- Which parameter passing mode for **v** below?

```
// remove first instance of target from vector v

void removeVal(int target,_____________________

  // find loc of target in v
  // if it's there,
  //    shift values to close up hole
  //    v.pop_back();
}
```

---

```
. . .
vector<int> v = readvals();
removeVal(32, v);
```

- What about letting client know whether target was found?

# Defining classes

- For basic stuff, mostly just syntactic differences between C++ and Java
- We'll look at `studentProg.cpp`
- and compare with the Java version we did earlier in the semester.

`Student.java`

`StudentTester.java`