# Introduction to C++ classes

Lecture 22

## Practical #1

- The first (of two) practical assignments.
  - The goal is to have you work on code in a way that is likely in future classes or the real world.
    - You get a specification, you are asked to write a program to do the task.
  - You may not ask for programming help.
    - GSIs and I will not answer coding questions related to the project.
    - We will answer specification questions

## Practical #1

- Project is to generate a histogram of grades.
  - We provide sample input and output values.
  - The project write-up explains how to use “diff” to compare your output to the right answer.
- Due this Sunday at noon
  - There is also an extra credit part that is probably a lot more work than it is worth.
    - But if the practical was easy…

## Coming Soon

- HW 4 due Wednesday 3/9 (noon)
- Practical 1 due Sunday 3/13 (noon)
- Quiz 2 on Wednesday 3/16 (in class)
  - Covers material since Exam 1.
- Project B due Friday 3/25 (9pm)
  - Assigned this Wednesday

## Plan

- Today:
  - C++ classes, an introduction
- Wednesday
  - Project B overview
  - C++ classes continued
- Friday (& probably Monday)
  - More on using C++ classes

## Classes/ Object oriented programming

- The basic idea is that as programs get large, they become difficult to organize and manage.
  - The traditional attack against this type of complexity is to group things into understandable blocks.
    - In fact, that is what we do when we use "functional decomposition". That is, breaking a problem into parts which we solve separately.

## More on OOP

- As programs get very large functional decomposition doesn't cut it.
  - The problem is there are *so* many functions you just have too much to keep track of at once.
  - So the basic idea is to organize things by data structures (structs, arrays, or whatever).
    - You then write functions which manipulate those data structures.

## Example: Complex numbers

- Say you are writing an electrical circuit simulation tool.
  - Such simulations involve complex numbers
- What you would like to do is write a bunch of functions that can perform the various complex number operations.
  - Say, add, subtract, divide, multiply, and exponentiation
- So in C (or Fortran or BASIC or…) what you would do is write a bunch of functions which do those operations
  - Say CpxAdd(a,b), CpxMult(a,b) etc.

## Without Classes

- By tradition (and generally good style) what we would do in C is:
  - Write the structure definition and all the relevant code in one file.
  - Ask that the user not manipulate our structure directly.
    - Instead they should use the supplied functions.
    - This allows us to potentially do some extra work to be sure all is well.
      - For example, checking array bounds.

```
const int SIZE=30;
int readA(int array[], int loc)
{
    if(loc>=0 && loc <SIZE)
        return(array[loc]);
    else
    {
        cerr << "Read access out of bounds\n";
        exit(1);
    }
}

void writeA(int array[], int loc, int value)
{
    if(loc>=0 && loc <SIZE)
        array[loc]=value;
    else
    {
        cerr << "Write access out of bounds\n";
        exit(1);
    }
}
```

## Problems?

- Well, for one, the functions are obviously bound to an array of a fixed size but…
  - There is really no association between the data structure and the functions
  - You'd have to go searching for those functions that work on that fixed size array.
- In general, you want the data structure and the functions tightly bound.

## Other problems

- Wonky naming
  - readA(), writeA()
    - Don't want these to be too big, too hard to type.
    - But readA and writeA are really not descriptive and someone else on the project might have used that name!
- Constant declared separate from functions
  - Just annoying.
    - Would like to group stuff together.

## Solution?

- Make functions and data elements all a member of the same group

```
#include<iostream>
using namespace std;

const int SIZE=30;
struct SIarray
{
    int array[SIZE];
    int read(int loc);
    void write(int loc, int value);
};
```

- Now functions are grouped with their data structure.
- Because functions are now members of the structure we can use names of “read” and “write”
- Const is still outside of structure for now. We will fix that later.

```
int SIarray::read(int loc)
{
    if(loc>=0 && loc <SIZE)
        return(array[loc]);
    else
    {
        cerr << "Read access out of bounds\n";
        exit(1);
    }
}

void SIarray::write(int loc, int value)
{
    if(loc>=0 && loc <SIZE)
        array[loc]=value;
    else
    {
        cerr << "Write access out of bounds\n";
        exit(1);
    }
}
```