Day 29 (Wed 04/06)

- review exercise 28
- day 29 recap questions
- exercise 29

Announcements/reminders

- submit "Midterm Project Contributions" survey on Gradescope by Friday at 11pm
  - do not assume late submissions will be accepted

- HW5 due *this evening* by 11pm

- HW6 due Wednesday 4/13 by 11pm
  - written assignment, late submissions not allowed

Exercise 28

```
// GradeList constructor
GradeList::GradeList(int capacity)
  : grades(new double[capacity])
  , capacity(capacity)
  , count(0) {
}
```

```
// GradeList add member function
void GradeList::add(double grade) {
  if (count >= capacity) {
    double *expanded =
      new double[capacity*2];
    for (int i = 0; i < count; i++) {
      expanded[i] = grades[i];
    }
    delete[] grades;
    grades = expanded;
    capacity *= 2;
  }
  grades[count++] = grade;
}
```

Exercise 28

```
// GradeList add (many) function
void GradeList::add(int howmany, double * grades) {
  for (int i = 0; i < howmany; i++) {
    add(grades[i]);
  }
}

// GradeList clear function
void GradeList::clear() {
  delete[] grades;
  grades = new double[1];
  capacity = 1;
  count = 0;
}
```

Exercise 28

Memory leak reported by valgrind:

```
==320==
==320== HEAP SUMMARY:
==320==     in use at exit: 64 bytes in 1 blocks
==320==   total heap usage: 9 allocs, 8 frees, 77,088 bytes allocated
==320==
==320== LEAK SUMMARY:
==320==    definitely lost: 64 bytes in 1 blocks
==320==    indirectly lost: 0 bytes in 0 blocks
==320==      possibly lost: 0 bytes in 0 blocks
==320==    still reachable: 0 bytes in 0 blocks
==320==         suppressed: 0 bytes in 0 blocks
==320== Rerun with --leak-check=full to see details of leaked memory
```

Exercise 28

Adding a destructor

```
// in grade_list.h (in the GradeList class definition)
~GradeList();

// in grade_list.cpp
GradeList::~GradeList() {
  delete[] grades;
}
```

Exercise 28

main2.cpp requires a default constructor

```
// in grade_list.h
GradeList();

// in grade_list.cpp
GradeList::GradeList()
  : grades(new double[1])
  , capacity(1)
  , count(0) {
}
```

Exercise 28

```
// begin() and end() functions can be defined in grade_list.h

double *begin() { return grades; }

double *end()   { return grades + count; }
```

Pointers can be used as iterators because they support the essential operations (dereferencing, advancing using ++, == and != to compare) required for iterator values.

1. What is overloading in C++?
2. Can you overload a function with the same name, same parameters, but different return type?
3. Is it true that we can overload all the operators of a class?
4. What is a copy constructor? When will it be called?
5. What happens if you don’t define a copy constructor?
6. What is the friend keyword? When do we use it?

1. What is overloading in C++?

Overloading means defining two functions (or member functions) with the same name.

This is allowed as long as the overloaded variants can be distinguished by number and/or types of parameters.

Note that you used overloading in Exercise 28: there were two "add" member functions in the GradeList class.

2. Can you overload a function with the same name, same parameters, but different return type?

No. Overloaded variants must be distinguishable by their argument(s).

3. Is it true that we can overload all the operators of a class?

Mostly. You can't overload the "." (member selection) or "::" (scope resolution) operators. All other operators may be overloaded.

4. What is a copy constructor? When will it be called?

A copy constructor initializes an object by copying data from another object of the same type.

E.g.:

```
std::string s("hello");

std::string s2(s);     // s2 initialized using std::string's copy constructor
std::string s3 = s;   // s3 initialized using std::string's copy constructor
```

The copy constructor is called any time an instance of a class needs to be initialized by copying an object of the same type. This includes passing an object to a function by value, and (maybe!) when returning an object from a function by value. (It's possible for the compiler to use "return value optimization" so that an object returned by value is constructed in the caller's stack frame, without the need for copying.)

5. What happens if you don’t define a copy constructor?

The compiler will generate a copy constructor automatically if one isn't explicitly defined.

The compiler-generated copy constructor will copy field values in order. (This is known as "member-wise" copying.)

Note: if the class has a non-trivial destructor (e.g., the destructor frees dynamic memory), member-wise copying in the copy constructor will result in serious program bugs. We'll discuss this on Friday.

6. What is the friend keyword? When do we use it?

The friend keyword allows a non-member function to be granted access to private members of a class.

It's *occasionally* useful for things like stream insertion and extraction operators (<< and >>), which can't be class members, but may need to use the internal data representation in an object.