

# 601.220 Intermediate Programming

Spring 2023, Day 29 (April 5th)

# Today's agenda

- Review exercise 28
- Day 29 recap questions
- Exercise 29

# Reminders/Announcements

- HW6 due Friday (April 7th) by 11 pm
  - Written homework, no late submissions
- HW7: will be released Friday evening, due 11 pm on Friday April 14th
- Final project team formation: soon

## Exercise 28 review


*// GradeList constructor*

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

} initialize list


## Exercise 28 review

```
// 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 review

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



delegation

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

## Exercise 28 review

Memory leak reported by valgrind:

```
==4874==
==4874== HEAP SUMMARY:
==4874==    in use at exit: 64 bytes in 1 blocks
==4874==   total heap usage: 9 allocs, 8 frees, 74,016 bytes allocated
==4874==
==4874== LEAK SUMMARY:
==4874==    definitely lost: 64 bytes in 1 blocks
==4874==    indirectly lost: 0 bytes in 0 blocks
==4874==    possibly lost: 0 bytes in 0 blocks
==4874==    still reachable: 0 bytes in 0 blocks
==4874==         suppressed: 0 bytes in 0 blocks
==4874== Rerun with --leak-check=full to see details of leaked memory
```

## Exercise 28 review

Adding a destructor:

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

```
~GradeList();
```

```
// in grade_list.cpp
```

```
GradeList::~GradeList() {  
    delete[] grades;  
}
```



## Exercise 28 review

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 review

*// 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 (dereference, advance using ++, == and != to compare) required for iterator values.

## Day 29 recap questions

- ❶ What is overloading in C++?
- ❷ Can you overload a function with the same name, same parameters, but different return type?
- ❸ Is it true that we can overload all the operators of a class?
- ❹ What is a copy constructor? When will it be called?
- ❺ What happens if you don't define a copy constructor?
- ❻ What is the `friend` keyword? When do we use it?

# 1. What is overloading in C++?

*Overloading* means define two or more functions (or member functions) with the same name.

This is allowed as long as the overloaded variables can be distinguished by number and/or types of parameters, or by `const`-ness.

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) or constness.

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

```
int n;  
cout << n << "\n";
```

↑  
left  
shift

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

`a.add(b)`

vs.

`a + b`

## 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.:

```
void f(string s) {  
    ;  
}
```

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

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

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?

```
Foo obj1;  
;  
Foo obj2(obj1);
```

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 deallocates dynamic memory), member-wise copying in the copy constructor will result in serious program bugs. We'll discuss this in a bit.

```
GradeList gl1;  
;  
GradeList gl2(gl1);
```





## 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 (<< and >>), which can't be class members, but may need to access the internal data representation of an object.

## Exercise 29

- `Complex` class to represent complex numbers
- Overload operators to do arithmetic
- Overloaded stream insertion operator (`<<`), as a friend function
- Talk to us if you have questions!

# Notes

# Notes

# Notes

# Notes

# Notes