6.S096 | January IAP 2013 | Undergraduate

Introduction To C And C++

Lectures and Assignments

C++ Introduction, Classes, and Templates

Lecture Notes

Lecture 5: C++ Introduction, Classes, and Templates (PDF)

Lab Exercises

There were no lab exercises to accompany this lecture.

Assignment 5

Problem 1

Now that we've transitioned from learning C to learning C++, we should be able to transition some C-style code that uses `struct`, `typedef`, and ordinary functions into C++ code that uses a single class to do the same job.

Download the C code below, and create a new file, `p1_grades.cpp.`

p1_grades (C)

Your job is to create a C++ class named `Grade` that has the same functionality as the old `struct Grade` and associated functions in the original C file. When you are finished, your C++ file should only have one function definition outside of the class: `main()`. You should use the following definition of main:

```
int main() {
        Grade g;
        int percent;

        printf("Enter two grades separated by a space. Use a percentage for
the first and letter for the second: ");
        scanf("%d", &percent;);
        scanf("\n");

        g.setByPercent(percent);
        g.print();

        g.setByLetter(getchar());
        g.print();

        return 0;
}
```

The names and interface of your Grade class should match the way the Grade instance is being used in the main function above. (That is, it should have member functions named `setByPercent`, etc. that are compatible with the use of those functions in `main`.)

Note the general way the grade program works: The user runs the program and is asked to enter two pieces of input. The first is an integer between 1 to 100 representing a percentage grade. The second input, separated on the command line by a space, is a letter grade (A, B, C, D, F). The output is two lines; each line shows the original and converted forms of the grade. So, for example, entering the input "`100 F`" would generate the lines:

```
Grade: 100: A
Grade: 45: F
```

The two input grades aren't related (a 100 isn't an F!). Instead, the inputs are used to show both directions of the conversion. Letter grades are converted to "nearby" percentages that fall into the right range. Don't worry about changing the grading logic. You can use the existing scale / system, including reusing `GRADE_MAP`.

Make sure your program compiles without warning, runs, and *definitely* use `valgrind` to ensure you have no memory leaks.

```
$ g++ -Wall p1_grades.cpp -o p1_grades
$ ./p1_grades
< provide a [percentageGrade] [letterGrade] input pair (like "97 D") >
<your test output>
```

Problem 2

In this problem, you will be converting a class that is specialized for integers into a templated class that can handle many types, including integers and structs. You will create a templated class named `List` that correctly initializes, manages, and de-allocates an array of a specified length. This is a nice class because the normal C arrays we've seen do not keep track of their length at runtime, but the class we're building will do that for us!

p2_templates (CPP)

When you're finished writing your templated `List` class, you should change your `main()` function to this code below (this is the same code that's in the starter file, p2_templates.cpp):

```
int main() {
    List<int> integers(10);
    for (int i = 0; i < integers.length; i++) {
        integers.set(i, i * 100);
        printf("%d ", integers.get(i));
    }
    printf("\n"); // this loop should print: 0 100 200 300 400 500 600 700 800 9

    List<Point *> points(5);
    for (int i = 0; i < points.length; i++) {
        Point *p = new Point;
        p->x = i * 10;
        p->y = i * 100;
        points.set(i, p);
        printf("(%d, %d) ", points.get(i)->x, points.get(i)->y);
        delete p;
    }
    printf("\n"); // this loop should print: (0, 0) (10, 100) (20, 200) (30, 300
}
```

This main function makes use of a typedef struct called Point. Here's its definition:

```
typedef struct Point_ {
    int x;
    int y;
} Point;
```

When run, your `main()` function should use the templated `List` class you've written yourself to produce this output:

```
0 100 200 300 400 500 600 700 800 900
(0, 0) (10, 100) (20, 200) (30, 300) (40, 400)
```

Naturally, this output should be generated by accessing the members of your templated `List` class, not by a hard-coded print statement.

To get you started, we've written a non-templated `IntList` class that just handles lists of integers:

```
class IntList {
	int * list;

public:
	int length;

	IntList(int len) {
		list = new int[len];
		length = len;
	}

	~IntList() {
		delete[] list;
	}

	int get(int index) {
		return list[index];
	}

	void set(int index, int val) {
		list[index] = val;
	}
};
```

You should use this class as a model for your own, templated `List` class, but you won't need IntList at all in the final code that you turn in, because you will have replaced it with your templated List class.

Make sure your program compiles without warning, runs, and *definitely* use `valgrind` to ensure you have no memory leaks.

```
$ g++ -Wall p2_templates.cpp -o p2_templates
$ ./p2_templates
<your test output>
```

Solutions

Solutions are not available for this assignment.