

Homework 4 --- Genericity Exercises (Code Polymorphism) --- CS 2704 Keller sections only.

Submit by Wednesday, May 3, 2000.

These two exercises involve practice using templates with functions.

Use the curator to submit your written solutions as either a Word or text document. Be sure to include your name, id number, and course section id in the submitted document.

Use the HW4 submission on the curator.

- (1) Write a templated function filter that filters elements from an STL list. The function should be templated by the type T of objects in the list, and the type of a function object that implements a predicate on the object type as an operator with signature `bool operator() (int)`. The function should remove all values from the list for which the predicate is true.

For instance, if we want to remove all of the odd numbers from a list of ints, we would define a function object

```
class OddInt {
public:
    bool operator() (int val) { return (val % 2 == 1);
};
```

and we could use it in a program like

```
void print_list(const list<int>& l) {
    list<int>::const_iterator l_i = l.begin();
    while (l_i != l.end())
        cout << *l_i++ << " ";
    cout << endl;
}
```

```
void main () {
    list<int> l;
    for (int i = 0; i < 20; i++) l.push_back(i);

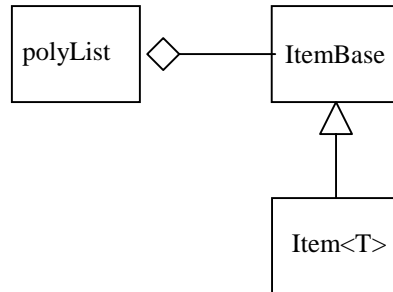
    print_list(l);
    filter(l,OddInt());
    print_list(l);
}
```

which would print

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 2 4 6 8 10 12 14 16 18
```

For your solution give the definition of the filter function. You should test it with a list that contains another type such as strings. In this case, you could use a function object that tests for objects with the letter 'a' as their first character.

- (2) One of the design patterns discussed in chapter 13 of the Reiss book is the idea of a wrapper class. We can combine the use of a wrapper class with an inheritance hierarchy to allow a polymorphic list without the (explicit) use of pointers. For instance, we can define classes with the following relationships



And, then write a program that inserts different kinds of objects into the list.

```
void main () {
    polyList l;
    l.insert(1); //insert an int
    l.insert("two"); //insert a string
}
```

Notice that I have not used pointers here.

- Give the declaration for the data field in polyList that is necessary to hold arbitrary items (use the STL list class to construct this type).
- Give the prototype necessary for the polyList::insert function so that this program would work. We shouldn't have to modify the function or add functions to also insert objects of other classes that we might create. (For fun, you can completely define the function, but this is not required.)
- OK, so you can get things into the list, but how do you get them out? Explain what you would have to do to extract an object from the list. You don't have to give code, but you should be specific about what would be necessary.