Templates in C++

Reading:

- Appropriate sections of your C++ book. E.g., Stroustrop chapter 13

A Simple ADT

Consider a new ADT called "holder".

DATA: A holder is either empty, or contains a single item.

OPERATIONS:

- store(o): If nothing is in the holder, store o there. Otherwise, overwrite what's in the holder with o, but only if o is smaller.
- print(): Prints the item in the holder.

As a first step, let's implement a holder that can only hold `int`s. Later, we will generalize it using templates.

A holder of ints

File holder.h

```
#ifndef HOLDER_H
#define HOLDER_H

#include <stdio.h>
#include <iostream.h>

class holder {
public:
     holder() { item = 0; }
     void store(int * x);
     void print();

private:
     int * item;
};

#endif HOLDER_H
```


File holder.cxx

```
#include "holder.h"

void holder::store(int * x){
  if (item == 0) {
      cout << "Storing first item in this holder\n";
      item = x;
  }
  else if (*x < *item) {
      cout << "Overwriting\n";
      item = x;
  } else {
      cout << "NOT overwriting\n";
  }
}


void holder::print() {
    cout << "Printing my item.\n";
    cout << *item << "\n";
}
```

File holderDriver.cxx

```
#include "holder.h"

void main() {
    holder * h = new holder();
    int i = 56;
    int j = 99;
    int k = -1321;
    h->store(&i);
    h->store(&j);
    h->store(&k);
    h->print();
}
```


Running the program

```
dvp.cs 122% g++ holder.cxx holderDriver.cxx

dvp.cs 123% a.out
Storing first item in this holder
NOT overwriting
Overwriting
Printing my item.
-1321
```


Generalizing with Templates

File holder.h

```
#ifndef HOLDER_H
#define HOLDER_H

#include <stdio.h>
#include <iostream.h>

template<class D> class holder {
public:
     holder() { item = 0; }
     void store(D * x);
     void print();

private:
     D * item;
};

#endif HOLDER_H
```


File holder.cxx

```
#include "holder.h"

template<class D> void holder<D>::store(D * x){
  if (item == 0) {
      cout << "Storing first item in this holder\n";
      item = x;
  }
  else if (x->compare(item)==-1) {
      cout << "Overwriting\n";
      item = x;
  } else {
      cout << "NOT overwriting\n";
  }
}

template<class D> void holder<D>::print() {
    cout << "Printing my item.\n";
    item->print();
}
```

File thing.h

```
#ifndef THING_H
#define THING_H

class thing {
public:
    thing(int n) { stuff = n; }
    int compare(thing * other);
    void print();

private:
    int stuff;
};

#endif THING_H
```


File thing.cxx

```
#include "thing.h"

int thing::compare(thing * other) {
    if (stuff < other->stuff)
            return -1;
    else if (stuff == other->stuff)
            return 0;
    else
            return 1;
}


// Later we'll learn how to define our own version
// of "<<" for 'thing's, so that one can say
//      thing * t = new thing(9182);
//      cout << t;
// rather than
//      t.print();

void thing::print() {
    cout << "My stuff is: " << stuff << "\n";
}
```


File holderDriver.cxx

```
#include "holder.h"
#include "thing.h"

void main() {
    holder<thing> * h = new holder<thing>();
    h->store(new thing(56));
    h->store(new thing(99));
    h->store(new thing(-1321));
    h->print();
}
```


Running the program

```
Storing first item in this holder
NOT overwriting
Overwriting
Printing my item.
My stuff is: -1321
```