CS 162
Intro to CS II

More Polymorphism and Templates

Polymorphism Revisited…

access.engr.orst.edu - PuTTY

```
 1 #include <iostream>
 2 #include "./Vehicle.h"
 3 #include "./bike.h"
 4 using std::cout;
 5 using std::endl;
 6
 7 //Only way to get the right toll without a pay
 8 //for each vehicle type!!!
 9 void pay(vehicle *v){
10    cout << v->get_toll() << endl;
11 }
12
13 int main(){
14    //vehicle v(4);  //Cannot make object of abstract class
15    bike b(1);
16    vehicle *vptr = &b;  //Polymorphism is late binding with pointer
17
18    pay(&b);
19
20    //v=b;  //upcasting is not polymorphism
21    //b=v;  //downcasting not advised
22
23    //cout << v.get_seats() << endl;
"main.cpp" 32L, 715C written                          17,0-1        Top
```

access.engr.orst.edu - PuTTY

```
 2 template <class T>
 3 class vector {
 4    private:
 5       T *v;
 6       int s;
 7    public:
 8       vector(){
 9          s=0;
10          v=NULL;
11       }
12       ~vector(){
13          delete [] v;
14       }
15       int size() {
16          return s;
17       }
18       T at(int i) {
19          return v[i];
20       }
21       void pushback(T ele) {
22          T *temp;
23          temp = new T[s];
24          for(int i=0; i<s-1; i++)
25             temp[i]=v[i];
26
27          delete [] v;
28          s++;
29          v = new T[s];
30          for(int i=0; i<s-1; i++)
31             v[i]=temp[i];
32
33          v[s-1]=ele;
34          delete [] temp;
35       }
36 };
```

access.engr.orst.edu - PuTTY

```
 1 #include <stdexcept>
 2 //#include <exception> //Don't need this when you include stdexcept
 3 #include <iostream>
 4 #include <string>
 5 #include <vector>
 6 #include "./vector.hpp"
 7 using std::cout;
 8 using std::endl;
 9 using std::string;
10 using std::exception;
11 using std::out_of_range;
12
13 int main(){
14    string s;
15    int n;
16    std::vector <int> v(2); //make a vector to hold 2 ints
17    vector <char> my_v;  //making my own vector
18
19    v[0]=1;
20    v[1]=2;
21    cout << v.size() << endl;
22    v.push_back(3);  //push the number 3 to the back of list
23    cout << v.size() << endl;
24
                                                    1,20          Top
```

Why Function Templates?

```
//at least C++ has overload
void swap(int &, int &);
void swap(char &, char &);
…
void swap(int &a, int &b){
   int temp = a;
   a=b;
   b=temp;
}
void swap(char &a, char &b) {
   char temp = a;
   a=b;
   b=temp;
}
```

Function Template…

```
//Have to have this header
template<class T>
void swap(T &, T &);
…
template<class T>
void swap(T &a, T &b){
   T temp = a;
   a=b;
   b=temp;
}
```

When can you get into trouble?

```
//Have to have this header
template<class T>
void func(T, T, int);
…
template<class T>
void func(T a[], T b[], int size){
  //a is already a reference
  //what if we wanted to swap values in arrays
}
```

Why make a class templates?

- What example have we seen so far?

Class Templates

```
//Have to have this header
template<class T>
class vector {
  public:
    vector();
    ~vector();
    void push_back(T);
  private:
    T *v;
};
```

Class Templates

```
//Have to have this header
vector::vector(){
   v=NULL;
}
vector::~vector(){
   delete [] v;
}
template<class T>
void push_back(T element){
   …
}
```