Recitation 06

Strings – I/Os – Templates

George Mappouras

10/20/2017

Classes in General (e.i. std::string)

- C++ offers a LOT of ready classes.
- Before writing complicated code use google to find and understand the existing classes.
- You do not need to “re-design the wheel”. Understand the methods that are defined in the class you are using.

std::string::max_size

std::string::push_back

std::string::replace

…and so on

Example1: Create a function that takes two strings and concatenates them together

std::string::append

http://www.cplusplus.com/reference/string/string/append/

Example1: Create a function that takes two strings and concatenates them together

```
std::string str = “My name is ”;
std::string str2 = “George”;
str.append(str2);
std::cout << str << ‘\n’;
```

Example2: Converting from C++ to C string and back

```
//From C to C++
const char *s = “Turn that to C++ string”;
std::string s_cpp(s);
//From C++ to C
char * s_c = new char[s_cpp.length()+1];
std::strcpy(s_c, s_cpp.c_str());
```

Templates

- A “variable” type
- Example: I want to create a class that would be an array of a type char, int, double etc. But I do NOT want to create unique classes for each type
- How can we declare variables of type T, where T can be any kind of type that I would later define in my program?

```
template<typename T>
```

Lets create a class of Arrays of any type - 1

Our class will have a pointer data to any type and a size variable that store the size of the array

```
template<typename T>
class MyArray{
private:
 T * data;
 size_t size;
public:
….
```

Lets create a class of Arrays of any type 2

We also want to make constructors.

Constructor1: should initialize to NULL and 0 if size is not specified. Constructor2: Should initialize to size if size is specified. Constructor3: Deep copy of a given Myarray array

```
public:
MyArray(): data(NULL), size(0){}
MyArray(size_t s): data(new T[s]), size(s){}
MyArray(const Myarray & p){
 size = p.size;
 data = new T[size];
 for (int i=0; i<size; i++)
   data[i] = p.data[i];
}
```

Lets create a class of Arrays of any type 2

We also want to make constructors. Constructor1: should initialize to NULL and 0 if no size is not specified. Constructor2: Should initialize to size if size is specified. Constructor3: Deep copy of a given Myarray array

```
public:
MyArray(): data(NULL), size(0){}
MyArray(size_t s): data(new T[s]), size(s){}
MyArray(const Myarray & p){
 size = p.size;
 data = new T[size];
 for (int i=0; i<size; i++)
   data[i] = p.data[i];
}
```

Lets create a class of Arrays of any type 3

Now a simple Distractor that will free the memory. Also a deep copy function overloading assign operator

```
~Myarray(){
  delete[] data;
}
Myarray& operator=(const Myarray& rhs){
  if(this!=&rhs){
    T * temp = new T[rhs.size];
     for (int i=0; i<rhs.size;i++)
        temp[i] = rhs.data[i];
    delete[] data;
    data = temp;
    size = rhs.size;}
}
```

How do I use a class with Templates

- We need to eventually inform the compiler of what type T means

```
MyArray<int> array_int(4);
MyArray<std::string> array_str(3);
```

How do I use a class with Templates

- We need to eventually inform the compiler of what type T means

```
MyArray<int> array_int(4);
MyArray<std::string> array_str(3);
```

Could I have said: MyArray<std::string> array_str(array_int); ?

How do I use a class with Templates

- We need to eventually inform the compiler of what type T means

```
MyArray<int> array_int(4);
MyArray<std::string> array_str(3);
```

Using copy constructor

Could I have said: MyArray<std::string> array_str(array_int); ?

How do I use a class with Templates

- We need to eventually inform the compiler of what type T means

```
MyArray<int> array_int(4);
MyArray<std::string> array_str(3);
```

Could I have said: MyArray<std::string> array_str(array_int); ?

NO types don’t much

Vectors and Pairs

std::vector<typename T>

std::pair<typename T1, typenameT2>

How can I create a vector of pairs of ints?

Vectors and Pairs

std::vector<typename T>

std::pair<typename T1, typenameT2>

How can I create a vector of pairs of ints?

Step1: vector<pair<typenameT1, typenameT2> >

Vectors and Pairs

std::vector<typename T>

std::pair<typename T1, typenameT2>

How can I create a vector of pairs of ints?

Step1: vector<pair<typenameT1, typenameT2> >

Step2: vector<pair<int, int> >

Vectors and Pairs

std::vector<typename T>

std::pair<typename T1, typenameT2>

How can I create a vector of pairs of ints?

Actual Code: std::vector<std::pair<int, int> >

Vectors and Pairs

std::vector<typename T>

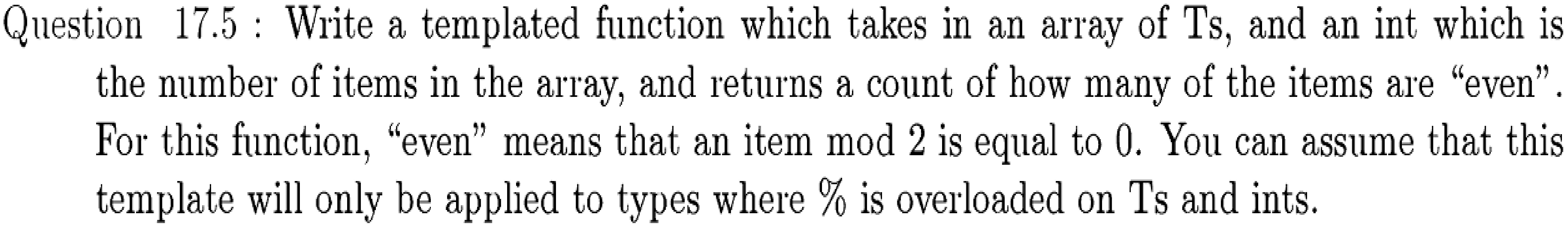
std::pair<typename T1, typenameT2>

How can I create a vector of pairs of ints?

Actual Code: std::vector<std::pair<int, int> >

Mind the gap!

Problem 1



Question 17.5 : Write a templated function which takes in an array of Ts, and an int which is the number of items in the array, and returns a count of how many of the items are “even”. For this function, “even” means that an item mod 2 is equal to 0. You can assume that this template will only be applied to types where % is overloaded on Ts and ints.

Answer 1

```
template<typename T>
unsigned countEven (T * array, size_t n){
  unsigned count = 0;
  for (size_t i=0; i<n; i++){
    if(array[i]%2 == 0){
      count++;
    }
  }
  return count;
}
```

Iterators

Usually when we want to iterate through the elements of an array we execute something like this:

```
for(int i=0; i<myObject.size(); i++){
   x=myObject[i];
}
```

Why that may be wrong? Think about Templates and Vectors

Iterators

We want “universal” iterators that can iterate through: iterator class
Lets take as example the std::vector

std::vector::begin() is defind as:
iterator begin(); //Points to the first element of the vector
std::vector::end() is defind as:
iterator end(); //Points past the last element of the vector

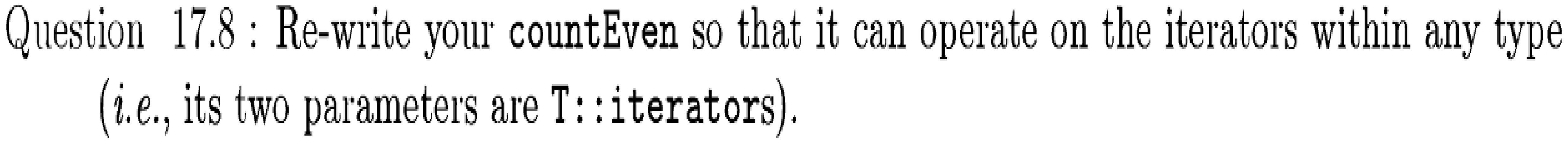
*it returns the corresponding element
++it points to the next element

Iterators

So lets re-write our for loop for vectors

```
std::vector<int> my_vector(100);
std::vector::iterator it = my_vector.begin();
while(it != my_vector.end()){
   x = *it;
   ++it;
}
```

Problem 2



Question 17.8 : Re-write your **`countEven`** so that it can operate on the iterators within any type (*i.e.*, its two parameters are `T::`**`iterators`**).

Answer 2

```
template<typename T>
unsigned countEven (typename T::const_iterator start, typename T::const_iterator end){
  unsigned count = 0;
  while( start != end){
    if((*start)%2 == 0){
      count++;
    }
     ++start;
  }
  return count;
}
```