VE280 Rescue Conference

Final review class

makersmelx

Lecture 21: Template & Container

Template

Key points

- Know why we use template (but may not be emphasized in the exam)
- Know how to write the class declaration
- Know how to use a template class (create an instance, use the STL container)

Template

Why we use template

- Only need to write TemplateList<T> instead of writing IntList and CharList respectively;
- Keyword: generic programming. Complex concept also used in Java, TypeScript, Python.
- Generic programming centers around the idea of abstracting from concrete, efficient algorithms to obtain generic algorithms that can be combined with different data representations to produce a wide variety of useful software.

—Musser, David R.; Stepanov, Alexander Complex., Generic Programming

Template

How to write declaration and use it?

- Practice:
  - Define a template class named CannonK with two template parameters T, U
    - Write its default constructor header
    - Write the header its copy constructor
    - Write its the header of its destructor
    - Write the header of “=“ overloading
    - Define a variable named *airport* that is a vector of CannonK with string and int

Container of pointers: 1 invariant & 3 rules

Understand why rather than just remember the words!

- **At-most-once invariant**: any object can be linked to at most one container at any time through pointer.
- **Existence**: An object must be dynamically allocated before a pointer to it is inserted.
- **Ownership**: Once a pointer to an object is inserted, that object becomes the property of the container. No one else may use or modify it in any way.
- **Conservation**: When a pointer is removed from a container, either the pointer must be inserted into some container, or its reference must be deleted.

Which functions in Project 4 have violations?

- `BinaryTree(Node *node);`

- `Node *find(const int &key) const;`

Polymorphic containers

Lab 6

```
int PluginManager::registerPlugin(CreateFunc *cFunc,
DestroyFunc *dFunc) {
    unique_ptr<Plugin, std::function<DestroyFunc>> p(cFunc(),
dFunc);
    pluginList.push_back(std::move(p));
    return 0;
}
```

- Why the main program of lab6 is able to load all kinds of plugins in its plugin list?
- If the create function of each plugin is a method of the plugin class, it is exactly the example.

Lecture 22: Operator Overload

Operator Overload

Key points

- Know how to declare an operator overload
- Know when you need to write a specific operator overload
- Know how to use friend keyword

Operator Overload

Binary Operators

```
Complex& Complex::operator= (const Complex& b);
Complex& Complex::operator+= (const Complex& b);
Complex& Complex::operator-= (const Complex& b);

Complex operator+ (const Complex& a, const Complex& b);
istream& operator>> (istream& is, Complex& b);
ostream& operator<< (ostream& os, const Complex& b);
```

- Note: How will the declaration be if the operator overload is written inside the class?

Operator Overload

Binary Operators

```
bool operator== (const Complex& a, const Complex& b);
bool operator!= (const Complex& a, const Complex& b);
bool operator< (const Complex& a, const Complex& b);
bool operator<= (const Complex& a, const Complex& b);
bool operator> (const Complex& a, const Complex& b);
bool operator>= (const Complex& a, const Complex& b);

template<class T>
const T& Complex::operator[] (size_t pos) const;

template<class T>
T& Complex::operator[] (size_t pos);
```

- Note: take notice of the const operator [] here

Operator Overload

Where to use?

- When you need to use to do I/O work with C++ style I/O functions
- When you are writing a template, but some types are missing the operations
- The above is also a limitation of the usage of template class that you must pay attention whether the operator/function used inside works for all types, namely works with template.

Lecture 23: Linear List and Stack

Linear and Stack

Key points

- Know the implementation of a linear list and a stack
  - How to insert and where to insert?
  - How to remove and which to remove?
  - How to access the data and which can be accessed?

Array vs. Linked List: Which is Better?

- They both have advantages and disadvantages

- Linked list
  - memory-efficient: a new item just needs extra constant amount of memory
  - not time-efficient for size operation

- Array
  - time-efficient for size operation
  - not memory-efficient: need to allocate a big enough array

Stack

Array vs. Linked list

- Please do not mistake this content by the direct comparison with array and linked list
- It is only about the implementation of stacks.
- For Linked List, you can also maintain an integer to record the size. The two implementation do not vary in time-efficiency.
- The two methods are just a trade-off between memory and time.

Array vs. Linked list

The general answer

- **An array is more memory-efficient than a Linked List**. The Linked List has to use extra memory to store the address of the next node
- **Insert and delete an element**: The linked list takes a constant time, while the array takes a linear time (with the number of elements).
- **Access a random element**: For an array it takes constant time by operator [], for a linked list, it takes a linear time (with the number of elements) for the worst case.

— it will be covered in VE281 and VE477

Stack

Application

- Parentheses Matching
- Web Browser Back
- Traversal the binary tree without recursion (I suggest you have a try on this)
  - Pre order
  - Post order
  - Mid order (no need to try this)

Lecture 24: Queue

Queue and Deque

Key points

- Know the implementation of a queue (Linked List, Array, Circular Array)
  - How to insert and where to insert?
  - How to remove and which to remove?
  - How to access the data and which can be accessed?
- Know the implementation of a deque (double Linked List, Circular Array) Project 5
  - How to insert and where to insert?
  - How to remove and which to remove?
  - How to access the data and which can be accessed?

Lecture 25 26: STL

STL Container

Key points

- Learn how to read docs in https://en.cppreference.com/w/
- Know how to use STL containers and when they can be used
  - Which header file is it placed?
  - How to get the size or see it is empty or not?
  - How to insert and where to insert?
  - How to remove and which to remove?
  - How to access the data and which can be accessed?
  - How to use the iterator?
  - How to use some tricks with STL?
  - ……..
  - Where and how should I ask for help if I forget the above things? (It happens to me very often)

STL Container

Iterator

- A generalization of pointer. Pointers to the elements of containers.
- Note that the address of the element the iterator points to is

```
&(*itr)
```

```
vector<int> v;// has some elements
int sum = 0;
vector<int>::iterator itr;
for(itr= v.begin(); itr != v.end(); ++itr)
   sum += *itr;

vector<int>::iterator itr;
itr = v.begin() + v.size()/2;
```

STL Container

Reference or Value

- Note that
  - operator [],
  - top(),
  - front(), back()…
- They all return the **reference** to the element in the container

STL Container

Tricks

```
for (int &i : x)
     sum += *itr;
```

It is the same as

```
for(itr= v.begin(); itr != v.end(); ++itr)
   sum += *itr;
```

Review Guidance

In my point of view

- Look through your P4 and P5 codes
- Understand and write codes to practice
- The emphasized points are the followings
  - Inheritance (virtual function and function override)
  - Memory management (deep copy, shallow copy, container of pointers)
  - template (declaration and implementation)

Some points

- Final paper of 2020 summer is not so…
- Never guess the standard rubric according to the points of this problem.
- For the correctness problem in the exam, please write things that you are sure about. Deduction will be applied if you point a right thing to be wrong.
- Prepare a correction tape in case you write something wrong for your writing code problem. Write clearly otherwise it will be hard for us to give you points.
- Do not hesitate to ask if you think the problem is not clearly described.
- We are NOT sure that we can understand what the instructor is thinking of.

Limitations

C++11

- **auto** keyword is not allowed in this exam
- **unique_ptr, shared_ptr** is not allowed in this exam

Good Luck…
Enjoy the summer break