Priority Queue

Queue with privilege

Nattee Niparnan

Intro

- Priority Queue is….
  - A queue with priority
  - Item with high priority is promoted to the front of the queue
    - There is no back of the queue
  - Priority is defined by having more value
    - Comparison, by default, is to use operator <, i.e., if item A < B is true, then B has higher priority
    - We can have custom comparator
- Has the same interface as stack

Example

For intuitive purpose only!

While the result is correct,

This is not really how

priority_queue work internally.

push(10)

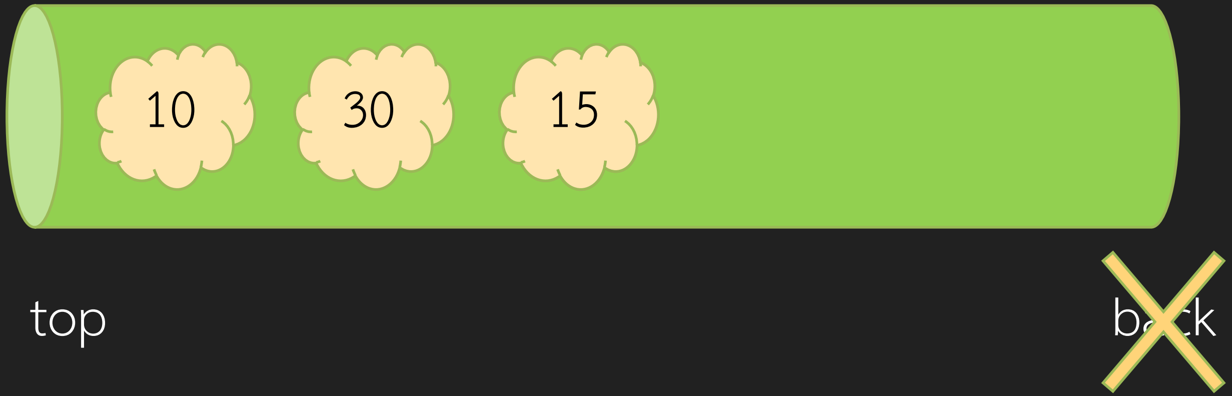
push(30)

push(20)

pop()

push(15)

pop()



Basic

size_t	q.size()
bool	q.empty()
void	q.push(T data)
void	q.pop()
T	q.top()

```cpp
#include <queue>
#include <iostream>

using namespace std;

int main() {
    priority_queue<int> pq;
    pq.push(10);
    pq.push(30);
    pq.push(20);
    cout << "Current size = " << pq.size() << " top is " << pq.top() << endl;
    pq.pop();
    pq.push(15);
    pq.pop();
    cout << "Current size = " << pq.size() << " top is " << pq.top() << endl;
}
```

Limitation

- Same limitation as stack, queue
  - No iterator
    - No begin(), end()
    - Can only access top of the priority_queue
    - If we wish to access all members, we have to pop it all
  - Do not call top(), pop() when the priority_queue is empty
- The data type must be comparable (similar to set and map)

Class in C++

I hope you have read the assignment

http://www.cplusplus.com/doc/tutorial/classes/

Quick Summary

- Syntax
  - class declaration must end with ;
  - Function definition can be outside the class
  - Access modifier is public:, private: protected:
  - constructor is a function with the same name of the class with no return type
- Object is a variable (instantiation) of a class
  - When declared, a constructor is called

Example 1

```cpp
#include <iostream>
#include <string>
using namespace std;

class Student {
public:
  void setFullname(string name,string surname) {
      this->name = name;
      this->surname = surname;
  }
  string getFullname() {
      return "[" + name + " " + surname + "]";
  }
private:
  string name,surname;
};

int main() {
    Student a;
    Student b;
    a.setFullname("nattee", "niparnan");
    cout << a.getFullname() << endl;
    cout << b.getFullname() << endl;
}
```

```cpp
#include <iostream>
#include <string>
using namespace std;

class Student {
public:
  void setFullname(string name,string surname);
  string getFullname();
private:
  string name,surname;
};

void Student::setFullname(string name,string surname) {
  this->name = name;
  this->surname = surname;
}

string Student::getFullname() {
  return "[" + name + " " + surname + "]";
}

int main() {
    Student a;
    Student b;
    a.setFullname("nattee", "niparnan");
    cout << a.getFullname() << endl;
    cout << b.getFullname() << endl;
}
```

Example 2: Constructor

```cpp
#include <iostream>
#include <string>
using namespace std;

class Student {
public:
  Student(float score) {   gpax = score;  }
  void setFullname(string name,string surname) {
      this->name = name;
      this->surname = surname;
  }
  string getFullname() { return "[" + name + " " + surname + "]"; }
  bool is1stHonor() { return gpax >= 3.6; }
private:
  string name,surname;
  float gpax;
};

int main() {
    Student a(2.95);
    a.setFullname("nattee", "niparnan");
    cout << a.getFullname() << endl;
    if (a.is1stHonor()) { cout << "YES" << endl; } else { cout << "NO" << endl;}
    // Student b; // <-- cannot compile because there is no default constructor
}
```

```
[nattee niparnan]
NO
```

Operator Overloading

How C++ has a function for each operator

Overview

- Let say we write a + b when a and b is an object of some classes.
  - This can be considered the same as calling a function plus(a,b)
  - C++ allow us to write a function for many operator and use it as an operator
  - For example we can write a function times(a,b) and let it be used as a * b
- This is call operator overloading

Example

```cpp
#include <queue>
#include <iostream>
#include <string>

using namespace std;

string operator*(string & lhs, const int & rhs) {
    string result = "";
    for (int i = 0;i < rhs;i++) {
        result = result + lhs;
    }
    return result;
}

int main() {
    string a = "abc ";
    cout << a * 3 << endl;
    //this gives "abc abc abc "
}
```

- Function must be named operator followed by the operator that we will overload
- Some operator takes two parameters (such as +, -, *, /, %)
- Some takes on (such as ++, --, !, *, &)

Using with data structure that require sorting

- We have seen several data structure that requires comparability of the data, such as set, map and priority queue
- If we want to use our class with these data structure, we need to tell them how can we compare a pair of them
- There are multiple ways to achieve this
    - Let us consider operator overloading

Overloading <

- As stated earlier, set, map and priority_queue use operator< to compare two elements
- It does not work if we overload operator>

```
1
attawith
```

```cpp
class Student {
public:
  Student(float score, string a, string b) {
    name = a;
    surname = b;
    gpax = score;
  }
  bool is1stHonor() { return gpax >= 3.6; }
  //not good, now our data is public
  string name,surname;
  float gpax;
  //overloading <
  bool operator<(const Student& other) const {
    return gpax < other.gpax;
  }
};

int main() {
  Student a(2.95,"nattee","niparnan");
  Student b(4.00,"attawith","sudsang");
  cout << (a < b) << endl;
  priority_queue<Student> pq;
  pq.push(a);
  pq.push(b);
  cout << pq.top().name << endl;
}
```

Custom Comparator

Why custom?

- By overloading operator<, we have defined default ordering of that class
- What if we need another ordering, just for this priority_queue only
  - For example, Student is ordered by gpax by default
  - What if we want our priority_queue to order by name instead, while keep the Student default ordering elsewhere
  - Better, can we have multiple priority_queue with different ordering?
- Can be done via comparator class

Example

```cpp
#include <iostream>
#include <string>
#include <queue>
using namespace std;

class Student {//same as before};

class StudentByNameComparator {
public:
  bool operator()(const Student& lhs,
                  const Student& rhs) {
      return lhs.name < rhs.name;
  }
};

class GpaxThenName {
public:
  bool operator()(const Student& lhs,
                  const Student& rhs) {
      if (lhs.gpax == rhs.gpax)
        return lhs.name < rhs.name;
      return lhs.gpax < rhs.gpax;
  }
};
```

```cpp
int main() {
  Student a(2.95,"nattee","niparnan");
  Student b(4.00,"attawith","sudsang");
  Student c(4.00,"vishnu","kotrajaras");
  cout << (a < b) << endl;
  StudentByNameComparator comp1;
  GpaxThenName comp2;
  priority_queue<Student,
                 vector<Student>,
                 StudentByNameComparator> pq(comp1);
  pq.push(a);
  pq.push(b);
  cout << pq.top().name << endl;

  priority_queue<Student,
                 vector<Student>,
                 GpaxThenName> pq2(comp2);
  pq2.push(a);
  pq2.push(b);
  pq2.push(c);
  cout << pq2.top().name << endl;

}
```

```
1
nattee
vishnu
```

Another Method, lambda-function

```cpp
#include <iostream>
#include <string>
#include <queue>
using namespace std;

int main() {
  auto compare = [](const string& lhs, const string& rhs) {
    return lhs.size() < rhs.size();
  };

  cout << "Result of compare function = " << compare("xxx","z") << endl;

  priority_queue<string,vector<string>,decltype(compare)> pq(compare);
  pq.push("somchai");
  pq.push("z");
  pq.push("abc");
  while (pq.empty() == false) {
    cout << pq.top() << endl;
    pq.pop();
  }

}
```

```
somchai
abc
z
```

- Compare is a variable of function type
- This one orders by length of string

Templating of priority_queue

- priority_queue requires 3 template parameters
- `priority_queue<T, Container = vector<T>, Compare = less<T>>`
- The first one is required (which is the type of the data
- The second and the third is optional (it has default type)
  - Second is the container (for now, just don’t think about it)
  - Third is the class for comparator (the class that we use to compare)
    - This one is default to `less<T>`

```
#include <iostream>
#include <string>
#include <queue>
using namespace std;

int main() {
  less<int> x;
  greater<int> y;

  int a = 10;
  int b = 3;
  cout << x(a,b) << endl;
  cout << y(a,b) << endl;
}
```

```
0
1
```

Using Comparator for set and map

- To use custom class with set and map, we need to do the same thing, let set and map know how to sort the data
  - Either make default ordering (overload<) in the custom class
  - Or use custom comparator when declare
- For set, the declaration is `set<T, Compare = less<T>>`
- For map, the declaration is `map<Key, T, Compare = less<Key>>`

Assignment

- Is any of vector<int>, set<int>, map<int,string>, queue<bool>, stack<vector<int>> comparable?
  - For any class that is “YES”, how it is ordered?
  - For example, if vector<int> is comparable, how {1,2,3} is compared to {1,2,3,4} or {2,3,4}

Short Summary

Data Structure Summary

Data Structure	Pro	Cons	Remark
pair<T1,T2>	Nothings... It just a pair of two data type		
vector<T>	• Fast access [ ] • Fast append	• Slow find • Slow insert, Slow Erase	
set<T>	• Fast find • Item is sorted	• Slower to just append data than vector, stack, queue • Iterate is also slow • Takes lots of memory	Require comparator
map<Key,T>			• Associative data type • Also require comparator of Key_Type
stack<T>	• Very fast push, pop	• Very limited functionality but has special uses	• No iterator • Order of data coming out depends on something (stack, queue depends on WHEN it is pushed, pq depends on value) • PQ requires comparator
queue<T>			
priority_queue <T>	• Fast get max • Fast delete max • Data is sorted • Memory efficient	• Slower to just append data than vector, stack, queue • Very limited functionality	

more data structure

- C++ has more data structure not really covered right now
  - list is a vector with faster insert / erase but does not have fast access
  - unordered_set, unordered_map are set and map that the data is not sorted but is much faster
  - deque (pronounced DECK) is a queue that can push, pop at both ends
  - multiset, multimap are set and map that allows duplicate entries