Programming in C/C++

Patrick Ho
peiqistar@gmail.com

(Day6)

Set

- a kind of associative container that stores unique elements, and in which the elements themselves are the *keys*. They're sorted.

- define a set variable
  set<*template_parameters*> variable_name
  e.g. set<int> mySet;

- To add data member, use insert()
  e.g. mySet.insert(10);

- To get the size of set, use size()
  e.g. size_t num=mySet.size();

- To find data member, use find()
  e.g. set<int>::iterator it = mySet.find(100);

- To access data member, using iterator

Queue

- a container to operate in a FIFO context (first-in first-out), where elements are inserted into one end and extracted from the other.

- define a queue variable
  queue<*template_parameters*> variable_name
  e.g. queue<double> myQ;

- To add in data member, using push() e.g. myQ.push(0.5);

- To get the size of queue, using size() e.g. size_t num=myQ.size();

- To access data member,
  a.using front() to get 1st member
  e.g. double dV = myQ.front();

  b.using pop() to remove 1st member
  c.using back() to get last member
  d.using empty() to check queue is empty or not

Stack

- a container to operate in a LIFO context (last-in first-out), where the element at the top is the one that was most recently added.

- define a stack variable
  stack<*template_parameters*> variable_name
  e.g. stack<int> myS;

- To add in data member, using push() e.g. myS.push(5);

- To get the size of stack, using size() e.g. size_t num=myS.size();

- To access data member,
  a.using top() to get last added member
  e.g. int iV = myQ.top();

  b.using pop() to remove top member
  c.using empty() to check stack is empty or not

Priority Queue

- a container, which is guaranteed that the top element is the largest element, based on the comparison function of object.

- define a priority_queue variable
  priority_queue<*template_parameters*> variable_name
  e.g. priority_queue<int> myPQ;

- To add in data member, using push() e.g. myPQ.push(5);

- To get the size of priority_queue, using size()
  e.g. size_t num=myPQ.size();

- To access data member,
  a.using top() to get 1st largest member
  e.g. double dV = myPQ.top();

  b.using pop() to remove 1st member
  c.using empty() to check queue is empty or not

List

- linked list sequence container class. It's good performance in insert/remove element operation.

- define a list variable
  list<*int*> variable_name
  e.g. list<int> myList;

- To add in data member, using push_back(), push_front(),insert()
  e.g. myList.push_back(5);

- To get the size of priority_queue, using size()
  e.g. size_t num=myList.size();

- To access data member, use iterator

- It has its own sort(), reverse(), clear(), empty() functions

Pair

- couples together a pair of values

- define a pair variable
  pair <*data_type1, data_type2*> variable_name
  e.g. pair<string, int> Person;

- To access data member, using *first* and *second*
  e.g. Person.first = “Patrick”;
  int iAge = Person.second;

- To make a pair, using make_pair(…) function, which is in <utility>
  e.g.
  Person = make_pair(“Patrick”, 50);

- It’s easy/better than define a class or your own data structure

BitSet

- special class for bit value operation
- Ask Google and study based on samples

STL Algorithm Functions

- find(): liner search
- swap(): swap 2 variable
- unique(): unify
- reverse(): reverse member order
- sort(): sorting
- lower_bound(): find a member iterator
- binary_search(): search a member
- min(), max():
- next_permutation(): rearranging element order (n!)

- Need clearly remember them first, then use it correctly

Problem Solving

Reorder

Some people stand up in order to take a picture, which needs the older person is in the middle. Please write a program to list all possible setting.

Input: data.in file. The 1st line is the total people number (2<=N<=10). 2nd to N+1 line has,

FirstName(string length<10) Age(integer)

e.g.

5
Tim 10
Tom 20
Jim 30
Joe 40
Jone 50

Output: each possible setting name list (sort in alpha order)

e. g.
Jim Joe Jone Tom Tim
Tim Jim Jone Joe Tom
Tim Joe Jone Jim Tom
Tim Tom Jone Joe Jim
Tom Jim Jone Joe Tim
Tom Joe Jone Jim Tim

Solution A:

Solve in class together

Using next_permutation, set