***DATA STRUCTURE***

1. ***Pair:***
2. Syntax:

Pair<type1, type2> v={value1, value2};

Tuple<type1, type2, type3….> v={….};

Cin>>p.first>>p.second; (valid)

1. Get value:

* Pair: v.first or v.second;
* Tuple: get<index>(v);

(save 1 couple number)

1. ***Set(multiset, unordered set):***
2. Syntax:

* Set: set<type> v;
* Multiset: multiset<type> v;
* Unordered\_set<type> v;

1. Function:

Set:

* Set save the elements in an unrepeatable way, the elements is unique.
* The elements in set have order, according ascending order.
* We should use it into count different elements, find, insert…. Follow situation.

Multiset:

* Like set, but multiset can save repeatable elements.

Unordered\_set:

* Like set, but it is unordered.

1. Manipulation with set:

* Set.insert(): insert into set 1 element.(input)
* Set<type> v(arr, arr+n); (input)(use arr pointer)(enter set elements from arr to arr+n)
* Set.size(): size of set.
* Set.empty(): test element in set, have or haven’t.
* Set.clear(): clear all elements in set.
* Set.count(key): check key is in set or not, if key have, return 1, else return 0;

Iterator:

* Set<type>::iterator it= set.begin();(similar with set.end(), set.rbegin(), set.rend())(a variable iterator it save index of set)

(We can access iterator it by dereference, it like 1 a pointer variable storage the address of set(\*Notes: it similar, we can’t access direct to it like pointer). )

* set.find(key): return a pointer to point index of key.
* Set.erase(): have 3 way to use:

Set.erase(rvalue): delete all rvalue in set, or multiset

Set.erase(iterator): delete value in iterator point to.

Set.erase(b\_iterator, e\_iterator): delete all value about (b\_it, e\_it).

* Advance(ptr, position): increment the iterator till position iterator= argument.

1. ***MAP, MULTIMAP, UNORDERED\_MAP:***

***(***liên quan đến tần suất suất hiện của các phần tử trong mảng***)***

1. Syntax:

Map<type1(key), type2(key value)> mp;

/\*Enter value in map:

mp[key]= value;

Or: mp.insert({key, value})

mp.insert(p);\*/

1. Operator in map:

* Map.Insert(pair) or insert({key, value}): insert into map positive order first->end.
* Map.count(x): count the occurrences of key(x) in map.
* Map.find(x): find the iterator of key(x) in map, first times.
* Map.size(): size of map.
* Map.erase(x): delete key x in map, delete a pair.
* Map.clear(): delete all elements in map.
* Map.upper\_bound(x): find a min element and this element > key x
* Map.lover\_bound(x): similar upper but this element>= key x.
* Map is accessed by map[key] and when we see map[key], we can understand that it is a value of key.

1. Multimap and unordered\_map:

Like set.

1. Notes:

* When we access to map with key is not declared, the value of key is 0.
* With multimap, let care of with map.erase(x) because it will delete every elements have key=x.

1. ***Stack:***
2. Syntax:

* Stack<type> variable;

1. Function:

* Like map, set… it is an array have LIFO type (last in first out).

1. Manipulation into stack:

* St.Empty() : test into stack. if it have elements, empty=true(!=0) else false(0);
* St.Size() : check a size of stack.
* St.push(x): push elements x into stack.
* St.pop(): delete last elements.
* St.top(): it is the last elements in stack.

1. ***Queue:***

FIFO : first in first out.

1. Syntax:

Queue<type> variable;

1. Manipulation:

* Push(): push in last of queue.
* Pop(): pop the first element be pushed in queue.
* Empty():
* Size():
* Front(): Like top, but here it is used to first element.

1. ***Deque****:*

(Double end queue)

1. Declaration:

#include<deque>

Deque<type> variable;

1. Manipulator:

* Size():
* Push\_front(): push “value” into front of queue
* Push\_back(): p ush “value” into end of queue
* Pop\_frond(): de lete “value” in frond of queue
* Pop\_back(): delete “value” in end of queue
* Empty():
* Back(): access element into end of queue
* Front(): like back() but into begin

\*Note: front is in right end else back or end is in left end.