An Introduction to MAPs*

Tutorial

(Ideas presented here may be used with Project 3)

* Originally prepared by Zubair in Winter 2003
Associative Containers

- Types of Containers:
  - Sequences [Vectors, Linked-List]
  - Associative [MAP, MultiMAP, HashTable]
  - Access to containers is managed by iterators

- provide fast keyed access to the objects in the container

- Constructed from:
  - Key Objects
  - Compare Functions
Associative Containers (continued …)

- Just like Vectors, Maps can also be thought of as arrays but here index can be other than integer.

```cpp
template <class K, class V> class Assoc{
public:
    V& operator[](const K&); //return reference to V corresponding to K
//...}
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td></td>
<td></td>
<td>V3</td>
<td>V4</td>
<td>V5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K1</th>
<th>K2</th>
<th>K3</th>
<th>K4</th>
<th>K5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>V3</td>
<td>V4</td>
<td>V5</td>
</tr>
</tbody>
</table>

AN ARRAY

A MAP
Where to use: For Large Dynamic Tables that require sequential or Random searching.
- Benefit: Quick Random Access
- Inefficient if keys not evenly distributed

- Based on Tree Data Structure
- MAP allows only unique keys
- Multimap allows similar keys
- Map is ordered, Hash_map if no order required
- Requires that less than operation < is implemented
Map implements association.

```cpp
Map<string, int> age;
age.insert("Zubair", 25);
age.insert("Shahid", 23);
...
...
string name = "Zubair";
int a = age.find(name);
```

AGE MAP

<table>
<thead>
<tr>
<th>KEY</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zubair</td>
<td>25</td>
</tr>
<tr>
<td>Shahid</td>
<td>21</td>
</tr>
<tr>
<td>Amna</td>
<td>23</td>
</tr>
<tr>
<td>Bilal</td>
<td>20</td>
</tr>
<tr>
<td>Sadaf</td>
<td>24</td>
</tr>
</tbody>
</table>

GUESS What is the value of a?
//This is a very simple Program to display maps at work

#include <iostream>
#include <map>
#include <string>

using namespace std;

int main(){
    map <string,int> age;
    age["Zubair"] = 25;
    age["Shahid"] = 21;
    age["Amna"] = 23;
    age["Bilal"] = 20;
    age["Sadaf"] = 24;
    cout<<age.size()<<endl;
    string a = "Zubair";
    cout<<"The age of "<<a<<" is 
"<<age[a]<<endl;
    return 0;
}
//CONTINUING WITH ABOVE EXAMPLE LETS PRINT THEM ALL
// A LOOP

map <string,int>::iterator i;
  //Just Like VECTORS we need iterators
  //Map supports both forward and backward iterator

for(i = age.begin(); i != age.end() ; i++ )
  //returns iterator to the start of MAP
  //return the element after last element NOTICE ! = is used not <
  //Increment the pointer

cout<<"The age of "<<(*i).first" is"<<(*i).second<<endl;
/*i is the iterator pointer, dereferencing it returns an object of type Pair (pair is defined in the utility class and is used by many containers).
Pair is <Key, Value> pair and
pair.first returns Key AND pair.second return value*/
  SEE CODE in Resources section of course website to see how pair is used
A bit more on pairs

//The Code of iteration follows this slide

- `pair<KeyType, ValueType>` is used with maps
  - Can use pairs outside of (inside) maps, too
- Can create `pair<K,V>` objects with `make_pair(k, v)`:
  ```cpp
  pair<string, int> si("foo", 42);
  si = make_pair("bar", 24);
  ```
```cpp
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main(){
    map <string,int> age;
    map <string,int>::iterator i;
    age["Zubair"] = 25;
    age["Shahid"] = 21;
    age["Amna"] = 23;
    age["Bilal"] = 20;
    age["Sadaf"] = 24;
    for(i = age.begin();i!=age.end();i++)
        cout<<"The age of"<<(*i).first<<"is"<<(*i).second;
    return 0;
}
```

Now the Code is:

Output is obvious
**MAP has a Unique KEY**

Map\<string, int\> age;
age.insert("Zubair", 25);
age.insert("Shahid", 21);
age.insert("Shahid", 27);
...
string name = "Shahid";
int a = age.find(name);

NOW What is the value of a?

- The size is now 4 and **last value** added for key "Shahid" is 27 hence `a = 27`
- NOTICE: VALUE IS REPLACED

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zubair</td>
<td>25</td>
</tr>
<tr>
<td>Shahid</td>
<td>21</td>
</tr>
<tr>
<td>Shahid</td>
<td>27</td>
</tr>
<tr>
<td>Amna</td>
<td>23</td>
</tr>
<tr>
<td>Bilal</td>
<td>20</td>
</tr>
</tbody>
</table>
To use subscript operator [] we must provide a default constructor. Its Behavior is:

- Perform a look up on keys
- if key is not found add the key and default value of mapped_value
- Else if key is found return the value of mapped_value in pair.

Due to default value constraint there is no [] for const maps

```cpp
void foo(){
  map<string,int>m; //map starting out empty
  int x = m["Henry"]; //Create New Entry for "Henry", initialize 0, return 0
  m["Harry"] = 7; // create a new entry for Harry, initialize 0 and return 7
  int y = m["Henry"]; // return the value for henry entry
  m["Harry"] = 9; // change the value from Harry's Entry to 9
}
```
Be Careful in Implementing Compare

- Map is a sorted container
- Needs to compare when *Inserting* and when *finding* (Think of the tree insertion and search)
- For sorting it needs *proper definition of compare*, when we apply it on objects other than basic types. (see my effort to write a more interesting example).
- By default comparison used for keys is < (less than). Therefore you may overload operator < ( ) in key_object.
- The other way is to pass a type that knows how to compare.

```cpp
struct compare{
    bool operator( ) (const Key_Type curr, const Key_Type other) const{
        // one example but you will know how your objects compare
        if(curr.Grade==other.Grade) return curr.ID<other.ID;
        else return curr.Grade<other.Grade;
    }
};
```

map < Key_Type, Value_Type, Compare > m;

compare is taken from my attempt to write an exciting example
Finding a key

- Since using [] will insert the key if it does not exist we use find.
- Also [] will overwrite the already inserted value so we will like to find before we insert.

```cpp
iterator find ( const key_type& k )

int main(){
    map <string,int> age;
    map <string,int>::iterator i;
    age["Zubair"] = 25;
    age["Amna"] = 23;
    string a = "Zubair";
    cout<<"The age of "<<a"" first time"<<age[a]<<endl;
    // Now lets put the test
    if(age.find(a) == age.end()){ //since key is present
        age["Zubair"] = 22; //Test Fails
    }
    cout<<"The age of "<<a"" second time"<<age[a]<<endl;
    return 0;
}
```
Some **MAP functions** you will commonly use:

- `m.insert(make_pair(key, value));` // inserts
- `m.count(key);` // times occurs (0, 1)
- `m.erase(key);` // removes it
- `m[key] = value;` // inserts it into the table
- `m[key]` // retrieves or **creates** a “default” for it
- `m.begin(), m.end()` // iterators
Where to Look for More information

- "The C++ Programming Language (Third Edition)" Chapter 17 section 4 by Bjarne Stroustrup
- Code available at class homepage (resources section)
How to Design

- Some ideas from Relational Algebra
- Types of relationship
  - One to One
  - One to Many
  - Many to Many
- Maps implement 1-1 relationship
- Multimaps implement 1-many relationship
- Many- Many can be simulated by two multimaps
A More Interesting Example [deals with most commonly used features]

If I find Time I will make code available
Multimap (if some one wants to use them)

• Like map, but 1:n instead
• Introduces new function:
  – mm.upper_bound(key) is an iterator that comes after all the elements specified
• No more [ ] indexing, either
• mm.count(key) will return count (0, 1, .. N)
Multimap example

```cpp
multimap<string, int> mm;
multimap<string, int>::iterator i = mm.find("EECS");
if (i != mm.end()) // we found some EECS
    for (; i != mm.upper_bound("EECS"); ++i)
        cout << i->second << endl;
```

Output:

```
314
430
345
```
Multimap example (continued)

```
// display all; could use display(mm)!
for (i = mm.begin(); i != mm.end(); )
{ const multimap<string, int>::iterator j = mm.upper_bound(i->first);
  cout << i->first << ' ' << mm.count(i->first) << "":";
  do
  { cout << i->second << " ";
    ++i;
  } while (i != j && i != mm.end());
  cout << endl;
}
```

Output:
EECS 3:314 430 345
PSCL 2:102 101