# CSCI 262
Data Structures

6 – Maps

## Introducing Maps

- **Map**
  - An abstract data type for associating **keys** with **values**
  - Keys must be unique, value can be anything
  - Similar to sets (and often built on them)
    - The map stores sets of **pairs** or **associations**
    - The pair first value is the key, determines uniqueness
  - Also known as a **Dictionary**
  - Also known as an **associative array**

## For the Mathematically Inclined

Mathematically, a map is a **partial function**
- Relates keys in one domain to values in another domain
- Each key maps to one and only one value
- However, values can be mapped to multiple keys
- Partial because we don’t map all possible keys

## Example

A map of strings to strings, storing words → definitions.

<table>
<thead>
<tr>
<th>Word</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Individual facts, statistics, or items of info</td>
</tr>
<tr>
<td>structure</td>
<td>anything composed of parts arranged together in some way; an organization</td>
</tr>
<tr>
<td>algorithm</td>
<td>a set of rules for solving a problem in a finite number of steps, as for finding the greatest common divisor</td>
</tr>
</tbody>
</table>
  ...

Definitions from Dictionary.com

## Example

Product database: a map of strings to tuples, storing SKUs (product id codes) → product descriptions, prices, etc:

<table>
<thead>
<tr>
<th>SKU</th>
<th>Description</th>
<th>Color</th>
<th>Price</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>427-WHT-100-A</td>
<td>Widgets, white</td>
<td>White</td>
<td>47.99</td>
<td>box</td>
</tr>
<tr>
<td>437-RED-100-A</td>
<td>Thingamajigs, red</td>
<td>Red</td>
<td>47.99</td>
<td>box</td>
</tr>
<tr>
<td>5190-FOO-66X</td>
<td>Misc. doodads</td>
<td>Black</td>
<td>12.49</td>
<td>pack of 6</td>
</tr>
</tbody>
</table>
  ...

Definitions from Dictionary.com
Types of Maps

Just like sets, we have two kinds:

- Ordered maps
  - Items are stored in key order, retrievable in key order
  - Keys must be comparable
  - Typically implemented using binary search trees

- Unordered maps
  - Items are stored in no particular order
  - Typically faster than ordered maps
  - Implemented using hash tables

The Map ADT

A Map does all of these efficiently:

- Get a value associated with a key (if in map)
- Put a key/value pair into map
- Remove a key/value pair from map
- Update the value associated with a key
- Determine if the map contains a key

STL Maps (Ordered)

```cpp
#include <map>
template <class K, class V> class map
```

Method summary:

- `at(K key)`
  - Get value associated with key; throws exception if not found
- `insert(pair<K,V> entry)`
  - Put a key/value pair into map
- `emplace(K key, V value)`
  - Put a key/value pair into the map
- `erase(K key)`
  - Remove key/value pair from map
- `find(K key)`
  - Get iterator to entry
- `count(K key)`
  - Count matching entries
- `size()`
  - Number of entries
- `empty()`
  - True if no entries
- `operator[](K key)`
  - Get and put and update (returns a reference to value associated with key; creates default entry if not found)

STL Maps Methods

There’s a lot to cover here.

We’ll dive into the more important methods in a moment.

First...

STL Pair

pair is an STL template class designed for one purpose: to hold two objects.

```cpp
#include <utility>
template <class A, class B> class pair
```

Public member variables (not methods):

```cpp
public member variables

- first  // first element
- second  // second element
```
Pair Usage - Creation
Verbose:
    pair<type1, type2> p;
    p.first = obj1;
    p.second = obj2;

Quicker:
    auto p = make_pair(obj1, obj2);

Sneaky quick way, when a pair is expected, e.g. as arg:
    { obj1, obj2 }

Pair Usage – Extracting Values
void foo(pair<type1, type2> p) {
    type1 a = p.first;
    type2 b = p.second;
    ...
}

Pair Example
void print_pair(pair<int, string> p) {
    cout << p.first << " : " << p.second << endl;
}

int main() {
    auto p1 = make_pair(17, "hello");
    print_pair(p1);
    print_pair( {42, "goodbye"} );
    return 0;
}

STL Map Methods – Getting
You can get values associated with a key several ways:
• at(key)
  • Returns a reference to value, if key exists in map
  • Throws exception if key not in map
• find(key)
  • Returns an iterator to key, value pair if in map
  • Returns end iterator otherwise (compare with .end())
• [key]
  • Always returns a reference to a value:
    • Value associated with key, if already in map
    • If not in map, creates entry in map using default for value!
    • Should not be used to test for containment!!!

STL Map Getting Example
map<string, string> m = {
    {"cat", "meow"},
    {"dog", "woof"}
};

cout << m["dog"];    // output: woof
cout << m.at("cat"); // output: meow
cout << m["frog"];   // output: (blank)
cout << m.at("turtle"); // exception!
STL Map – Testing for Containment

Verbose but fast:
```cpp
map<string, string>::iterator it = m.find("bunny");
if (it != m.end()) {
    cout << it->second << endl;
} else {
    cout << "No bunny!" << endl;
}
```

Less verbose, but no access to iterator:
```cpp
if (m.find("bunny") != m.end()) { ... }
or:
if (m.count("bunny") > 0) { ... }
```

Map Example
```
map<string, int> lengths;
lengths.emplace("apple", 5);
lengths.emplace("orange", 6);
lengths.emplace("pear", 4);
```
```
cout << "Ask me a word: " << endl;
string s;
cin >> s;
if (lengths.count(s) > 0) {
    cout << s << " has " << lengths[s] << " letters." << endl;
} else {
    cout << "I do not know the word " << s << ".";
    cout << endl;
}
```

STL Map - Putting

Again, several choices:
- `m.insert( { "snake", "hiss" } )`
  - Parameter is a pair object
  - Will not overwrite/update existing entry
- `m.emplace("snake", "hiss")`
  - More flexible parameters – pair or key, value
  - Will not overwrite/update existing entry
- `m["snake"] = "hiss"`
  - Will overwrite/update existing entry

Another Map Example
```
map<string, int> frequencies;
ifstream fin("text.txt");
while (!fin.eof()) {
    string w;
    fin >> w;
    frequencies[w]++;
}
```
```
for (auto entry: frequencies) {
    cout << entry.first << " has " << entry.second << " occurrences." << endl;
}
```

Performance Note
- Finding an entry in a map: very fast
- Getting/putting an entry: depends on size of key and value (copies are made!!!)
  - E.g.  
    ```cpp
    map<int, vector<string>> m;
    // fill m with entries, with long vector values
    vector<string> v = m[42];   // copy made
    v.push_back("expensive");  // copy made
    m[42] = v;                  // copy made
    // better alternative
    m[42].push_back("quick");  // no copies made
    ```

Up Next
- **Wednesday, Jan. 31**
  - Debugging
  - Reading: Chapter 12.1
- **Friday, Feb. 2**
  - Lab 4 – Sets & Maps
  - APT 2 due
  - Next assignment (TBD)