1. The Class pair
   - Frequency tables
   - A vector of pairs

2. Sets, Multisets, and Maps
   - Using a set
   - Using a multiset
   - Frequency table with a map

3. Set Functions
   - to_string() and read_set()
   - Membership, union, difference, and intersection

MCS 360 Lecture 27
Introduction to Data Structures
Jan Verschelde, 30 October 2017
Pairs, Sets, and Maps

1. the Class **pair**
   - frequency tables
   - a vector of pairs

2. Sets, Multisets, and Maps
   - using a set
   - using a multiset
   - frequency table with a map

3. Set Functions
   - to_string() and read_set()
   - membership, union, difference, and intersection
A frequency table counts the number of occurrences.

If the objects we count are limited in number, then we can use an array.

But suppose:
- unknown in advance how many objects;
- fast access needed to the frequency table.

We use the frequency table application as a running example to illustrate *associative containers*. 
character frequencies

Our running main program is

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

int main()
{
    do
    {
        cout << "give character (dot . to stop) : ";
        char c; cin >> c;
        if(c == '.') break;
        // update frequency table
    }
    while(true);
    return 0;
}
```

Introduction to Data Structures (MCS 360)
1. **the Class** `pair`  
   - frequency tables  
   - a vector of pairs

2. **Sets, Multisets, and Maps**  
   - using a set  
   - using a multiset  
   - frequency table with a map

3. **Set Functions**  
   - `to_string()` and `read_set()`  
   - membership, union, difference, and intersection
If we include `<utility>`, then we can work with pairs:

```cpp
pair<char, int> p;
p.first = c;
p.second = 1;
```

The first member of our pair is of type `char`.
The second member of our pair is of type `int`.

The frequency table is a vector of pairs:

```cpp
vector<pair<char, int>> f;
```

For a pair `p` as above, do: `f.push_back(p)` at the first occurrence of `c`. 
int get_index ( vector< pair<char,int> > f, char c );

// returns -1 if c does not belong to f,
// otherwise returns the index of c in f

int get_index ( vector< pair<char,int> > f, char c )
{
    for(int i=0; i < f.size(); i++)
        if(f[i].first == c) return i;

    return -1;
}
int k = get_index(f,c);
if(k > -1)
    f[k].second++;
else
{
    pair<char,int> p;
    p.first = c;
    p.second = 1;
    f.push_back(p);
}

void write ( vector< pair<char,int> > f )
{
    for(int i=0; i<f.size(); i++)
        cout << f[i].first << " occurred "
            << f[i].second << " times " << endl;
}
1. The Class `pair`
   - frequency tables
   - a vector of pairs

2. Sets, Multisets, and Maps
   - using a set
   - using a multiset
   - frequency table with a map

3. Set Functions
   - `to_string()` and `read_set()`
   - membership, union, difference, and intersection
the Abstract Data Type of a set

abstract <typename T> set;
/* A set is an unordered collection of items of type T. */

abstract bool empty ( set s );
postcondition: empty(s)
    == true if s is empty,
    == false if s is not empty;

abstract size_t size ( set s );
postcondition: size(s) is the number of elements;

abstract bool member ( set s, T item );
postcondition: member(s, item)
    == true if item is in s,
    == false if item is not in s;

abstract void insert ( set s, T item );
postcondition: after insert(s, item): member(s, item) is true;

abstract void remove ( set s, T item );
precondition: member(s, item);
postcondition: after remove(s, item): member(s, item) is false;
For any two sets $S_1$, $S_2$, we can take the union $S_1 \cup S_2$, the intersection $S_1 \cap S_2$, and the difference $S_1 \setminus S_2$.

abstract set union ( set s1, set s2 );
postcondition: for s = union(s1, s2), we have:
  for all items, if member(s, item),
  then member(s1, item) or member(s2 item);

abstract set intersection ( set s1, set s2);
postcondition: for s = intersection(s1, s2), we have:
  for all items, if member(s, item),
  then member(s1, item) and member(s2, item);

abstract set difference ( set s1, set s2 );
postcondition: for s = difference(s1, s2), we have:
  for all items, if member(s, item),
  then member(s1, item) and not member(s2, item);
sets are iterable – ADT continued

abstract iterator begin ( set s );
precondition: not empty(s);
postcondition: begin(s) refers to the first item of s;

abstract iterator end ( set s );
precondition: not empty(s);
postcondition: end(s) refers to one past the last item of s;

abstract void advance ( set s, iterator i );
precondition: i is not equal to end(s);
postcondition: after advance(s, i), i refers to the next item of s;

abstract T dereference ( set s, iterator i );
precondition: i is not equal to end(s);
postcondition: dereference(s, i) is the item i refers to in s;
using a set

The STL class set is used as

```cpp
set<char> S;
```

Processing a character `c`:

```cpp
pair<set<char>::iterator, bool> p = S.insert(c);
if(p.second)
    cout << " occurred for the first time";
else
    cout << " already occurred: " << *p.first;
```

Calling `S.insert(c)` returns a pair `p`:

1. `p.first` indicates where `c` occurs in `S`;
2. `p.second` is true if `c \notin S` before insert.
void write ( set<char> S );
// writes the set of characters

void write ( set<char> S )
{
    for(set<char>::const_iterator i=S.begin();
        i != S.end(); i++)
        cout << " " << *i;
}

The elements in the set are written in order,
that is: a b c appears even as inserted in other order.
reversing the order

By default, the second argument when instantiating a set is the natural order induced by the < (less than) operator.

To reverse the order:

```cpp
struct reverse_compare
{
    bool operator()(const char &a, const char &b) const
    {
        return (a>b);
    }
};

int main()
{
    set<char, reverse_compare> R;
}```
different orders

The order on a set $S$ must be a total order $\prec$:

- for every $a, b \in S$: either $a \prec b$ or not;
- if not $a \prec b$ and not $b \succ a$, then $a = b$;
- if $a \prec b$ and $b \prec c$, then $a \prec c$.

Often an order is not unique, consider support sets, e.g.:

$$A = \{ (4, 2), (2, 2), (2, 1), (1, 5) \}$$

$A$ collects the exponents of a polynomial $f$, exponents corresponding to nonzero coefficients.

We can write $f$ in a *pure lexicographic* order:

$$f = c_{4,2}x^4y^2 + c_{2,2}x^2y^2 + c_{2,1}x^2y + c_{1,5}xy^5$$

or in a *degree lexicographic* order:

$$f = c_{4,2}x^4y^2 + c_{1,5}xy^5 + c_{2,2}x^2y^2 + c_{2,1}x^2y.$$
vectors versus sets

Characteristics of a vector:
- items are stored consecutively as an array;
- subscripting operator \([ \) \] for direct access.
→ vector is encapsulation of the C array type

Characteristics of a set:
- location where the items are stored is transparent;
- iterator follows order defined by function object;
- membership test with `find` method.
→ set is stored as balanced search tree
Pairs, Sets, and Maps

1. the Class `pair`
   - frequency tables
   - a vector of pairs

2. Sets, Multisets, and Maps
   - using a set
   - using a multiset
   - frequency table with a map

3. Set Functions
   - `to_string()` and `read_set()`
   - membership, union, difference, and intersection
using a multiset

The STL contains a `multiset`, instantiated as

```cpp
multiset<char> S;
```

In a multiset duplicate characters may occur.

Processing a character `c`:

```cpp
M.insert(c);
```

Iterating over the elements of the multiset, the elements are sorted so we count their occurrences.
writing frequency table

```cpp
void write_frequencies ( multiset<char> S )
{
    int count = 0;
    char current;

    for(multiset<char>::const_iterator i=S.begin(); i != S.end(); i++)
    {
        if(i == S.begin())
        {
            current = *i;
            count = 1;
        }
        else
        {
            current = *i;
            count += 1;
        }
    }
}
```
else
{
  if(*i == current)
    count++;
  else {
    cout << current << " occurs "
         << count << " times" << endl;
    count = 1;
    current = *i;
  }
}
} // end for
if(count > 0)
  cout << current << " occurs "
       << count << " times" << endl;
1. **the Class pair**
   - frequency tables
   - a vector of pairs

2. **Sets, Multisets, and Maps**
   - using a set
   - using a multiset
   - frequency table with a map

3. **Set Functions**
   - to_string() and read_set()
   - membership, union, difference, and intersection
maps as sets of pairs

A map is a set of ordered pairs, e.g.:

\[
\{(a, 1), (c, 3), (b, 3), \ldots\},
\]

where the types of the pair are

1. an index type for unique keys;
2. a value type which need not be unique.

This is an *onto* mapping, different keys are associated with values that may be the same.

Like a vector, a map has a subscripting operator \[\]. We could view a vector is a special map with `size_t` as index type. The map is known as an *associative array*. 

With a multimap, the key does not need to be unique. Example: `multimap<string, int>` for an index to word and page number.
#include <iostream>
#include <map>
using namespace std;

int main()
{
    map<char, int> M;
    do
    {
        cout << "give character (dot . to stop) : ";
        char c; cin >> c;
        if(c == '.') break;
        if(M.find(c) == M.end())
        {
            M[c] = 1;
        }
        else
        {
            M[c]++;
        }
    }
    while(true);
    return 0;
}
writing the map

```cpp
void write( map<char,int> M );
// writes the frequency table

... happens again with an iterator ...

void write( map<char,int> M )
{
    for(map<char,int>::const_iterator i = M.begin();
        i != M.end(); i++)
    {
        cout << i->first << " occurs " << i->second << " times" << endl;
    }
}
Pairs, Sets, and Maps

1. The Class `pair`
   - frequency tables
   - a vector of pairs

2. Sets, Multisets, and Maps
   - using a set
   - using a multiset
   - frequency table with a map

3. Set Functions
   - `to_string()` and `read_set()`
   - membership, union, difference, and intersection
function to_string()

string to_string ( set<char> S )
{
    ostringstream r;
    bool first_time = true;
    r << "{";
    for(set<char>::const_iterator i=S.begin();
         i != S.end(); i++)
    {
        if(first_time)
            first_time = false;
        else
            r << ",";
        r << " " << *i;
    }
    r << " "}
    return r.str();
}
function read_set()
{
    set<char> S;

    do
    {
        cout << "give character (dot . to stop) : ";
        char c; cin >> c;
        if(c == '.') break;
        S.insert(c);
    }
    while(true);

    return S;
}
Pairs, Sets, and Maps

1. The Class `pair`
   - frequency tables
   - a vector of pairs

2. Sets, Multisets, and Maps
   - using a set
   - using a multiset
   - frequency table with a map

3. Set Functions
   - `to_string()` and `read_set()`
   - membership, union, difference, and intersection
the `find()` method

```cpp
set<char> random_set ( int n );
// returns a random set of n characters

int main()
{
    cout << "give number of elements : ";
    int n; cin >> n;

    set<char> A = random_set(n);
    cout << "give a character : ";
    char c; cin >> c;

    set<char>::iterator i = A.find(c);
    if(i == A.end())
        cout << c << " does not occur in A" << endl;
    else
        cout << c << " does occurs in A as "
        << *i << endl;
```
defining a subset

$ /tmp/setmem
give number of elements : 10
the set A : { b, c, d, q, t, x, y } 
give a character : d
d does occurs in A as d
defining a subset ...
-> give start character in A : d
-> give stop character in A : t
the subset { d, q, t }
lower_bound and upper_bound

cout << "defining a subset ...
" << endl;
cout << "-> give start character in A : ";
char start; cin >> start;
cout << "-> give stop character in A : ";
char stop; cin >> stop;

set<char> B;
for(set<char>::const_iterator
   i = A.lower_bound(start);
i != A.upper_bound(stop); i++) B.insert(*i);
cout << "the subset " << to_string(B) << endl;

lower_bound returns iterator to the first occurrence
upper_bound returns iterator to smallest larger item
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;

int main()
{
    set<char> A, B, U, D, I;

    A = read_set();
    cout << "the set A : " << to_string(A) << endl;
    B = read_set();
    cout << "the set B : " << to_string(B) << endl;
union with adapter

code continued...

    set_union(A.begin(),A.end(),
               B.begin(),B.end(),
               inserter(U,U.begin()));
    cout << "union of A and B : "
         << to_string(U) << endl;

The inserter is an iterator adapter which inserts elements into a container.
The arguments of inserter are
  • the container where to do the insert,
  • an iterator within the container.
Each insertion appends to the current end of the container.
difference and intersection

computing difference and intersection is similar:

```cpp
def set_difference(A.begin(), A.end(),
                B.begin(), B.end(),
                inserter(D, D.begin()));
def set_intersection(A.begin(), A.end(),
                    B.begin(), B.end(),
                    inserter(I, I.begin()));```

cout << "difference of A and B : "
   << to_string(D) << endl;

cout << "intersection of A and B : "
   << to_string(I) << endl;
```
Summary + Exercises

Started chapter 9 on STL sets and maps.

Exercises:

1. Define a templated class `triplet` to store three items of the types given in the templates. Provide a constructor and members `first`, `second`, and `third`. Test the `triplet` with interactive code to store a set of names, as family, given, and middle name.

2. Consider set of `vector<int>` objects. Give code for function objects `purelex` and `deglex` to define the `<` operator respectively for pure lexicographic and degree lexicographic order. Show how to convert a `set<vector<int>, purelex>` object into a `set<vector<int>, deglex>` object.
Given two sets $X$ and $Y$, the product $X \times Y$ is defined as 
\[ \{ (x, y) \mid x \in X, y \in Y \} \]. Overload the operator $\ast$ to take the product of a set of `char` with a set of `int` objects. The result is stored as a set of pairs.

Redo the previous exercise so you can make the product of sets of any type.