Object Oriented Programming I

C++ Standard Template Library (STL):

- Containers
- Iterators
- Algorithms
- Strings

STL Containers

- An STL container is a collection of values of the same type
- But for this we have arrays and vectors already. Isn’t this enough?
- No!
- Vectors correspond to contiguous memory blocks in the RAM: value1, value2, value3,…
- This makes many operations with vectors inefficient

Example: Concatenate Two Vectors

- The entries of vectors are stored in contiguous memory blocks.
- Concatenating vectors requires copying all entries into new memory blocks.
- Can be very inefficient.
- For concatenation, lists are much more efficient

Problem 1

- Measure the time which is needed to concatenate two vectors of of length 1000,000 with entries of type double
- To concatenate vectors **v**, **w**, either
  - create a new vector **z** of double length and assign the entries ov **v**, **w** to **z** or
  - resize **v** and assign the entries of w to the new entries of **v**
- For time measurement, see next slide

Time Measurement

- Needs #**include<ctime>**
- **clock()** returns the number of “clock ticks” elapsed since program started
- Constant **CLOCKS_PER_SEC** gives the number of clock ticks per second
- Code snippet:

```
int start, end;
start=clock();
// commands whose execution time is to be measured
end =clock();
double t = (double)(end-start)/CLOCKS_PER_SEC;
// t is the time in seconds needed
```

Overview of STL Containers

- Most commonly used are vector, list, set, queue, priority_queue, stack
- Each of them is very good in certain situations and very bad in others
- For lists:
  - very good when elements have to be inserted and removed and lists have to be combined.
  - very bad when it is necessary to access and manipulate elements (no access by indices possible!)

Choice of Container

- By default, use vector. If there are obvious inefficiencies, consider other choices
- If elements are often inserted or removed, choose list.
- However, lists only make sense if no direct access to elements (by index) is necessary.
- If you often need the search for elements with certain properties, use set.
- Again, this only makes sense if no direct access to elements is necessary.
- Use queues and stacks if required by algorithms.

STL Lists

Lists

- Need **#include<list>**
- The elements of list are stored like in a chain a-b-…-x-y.
- The first element (here a) is the front, the last element (here y) is the back.
- An empty list is created with **list<type> ListName** *;*
- The type is the type of the elements of the list.
- All elements equal to **e** are removed with **ListName.remove(e);**

Basic Operations with List L

- Insert an element at the front: **L.push_front(e);**
- Insert and element at the back: **L.push_back(e);**
- Access element at front: **L.front();**
- Access element at end: **L.back();**
- Remove element from the front: **L.pop_front();**
- Remove element from the back: **L.pop_back();**
- Size of list: **L.size()**
- Append list L2 to list L1: **L1.splice(L1.end(),L2);**
  (after this L1 will contain all elements and L2 will be empty)
- For other functions for lists, see
  http://www.cplusplus.com/reference/stl/list/

Problem 2

- Create a list L1 containing the numbers 0,1,…,9
- Create a list L2 containing 10,…,19
- Append L2 to the end of L1
- Print the sizes of L1 and L2 to the screen
- Remove the even numbers from L1
- Print the first and last element of L1 to the screen

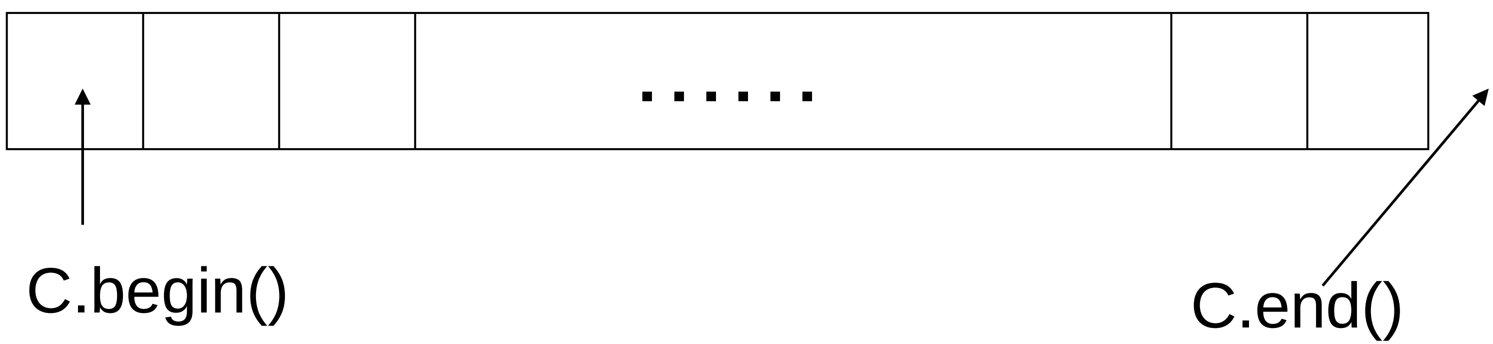
Problem 3

- Measure the time which is needed to concatenate two lists of of length 1000,000 with entries of type double and compare the result with Problem 1
- For the concatenation of list, use the splice function

STL Iterators

STL Iterators

- Iterators are used to iterate over containers.
- The value of an iterator **I** is a position in the container and ***I** is the value at this postion.
- **C.begin()** is the position of the first element of a container **C.**
- **C.end()** is the position directly after (!!!) the last element of **C.**



STL Iterators (cont.)

- For using **iterators**, you need **#include<iterator>**
- Iterator declaration:
  container<type>::iterator IteratorName;
- Examples:
  list<int>::iterator I;
  set<double>::iterator I;
- Going to then next position:
  IteratorName++;
- Going to the last position:
  IteratorName--;
- Accessing the element at the current position:
  ***IteratorName**

Example

```
#include <cstdlib>
#include <iostream>
#include <vector>
using namespace std;

int main(int argc, char *argv[])
{
    vector<int> v(10);
    for(int i=0;i<10;i++)
        v[i]=i;

    vector<int>::iterator I;
    I=v.begin();  // postion of I is first element
    cout << "first element: " << *I << endl;
    I++;    // go to next element
    cout << "next element: " << *I << endl;
    I+=5;  // go 5 elements further
    cout << "7th element: " << *I << endl;
    I=v.end();  // position of I is now BEYOND last element
    cout << *I << endl; // error, value undefined
    I--;          // position is now the last element
    cout << "last element: " << *I << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Iterator Offset

- To determine how many elements an iterator **I** is offset from the first element of a container **C** use **int(I-C.begin())**

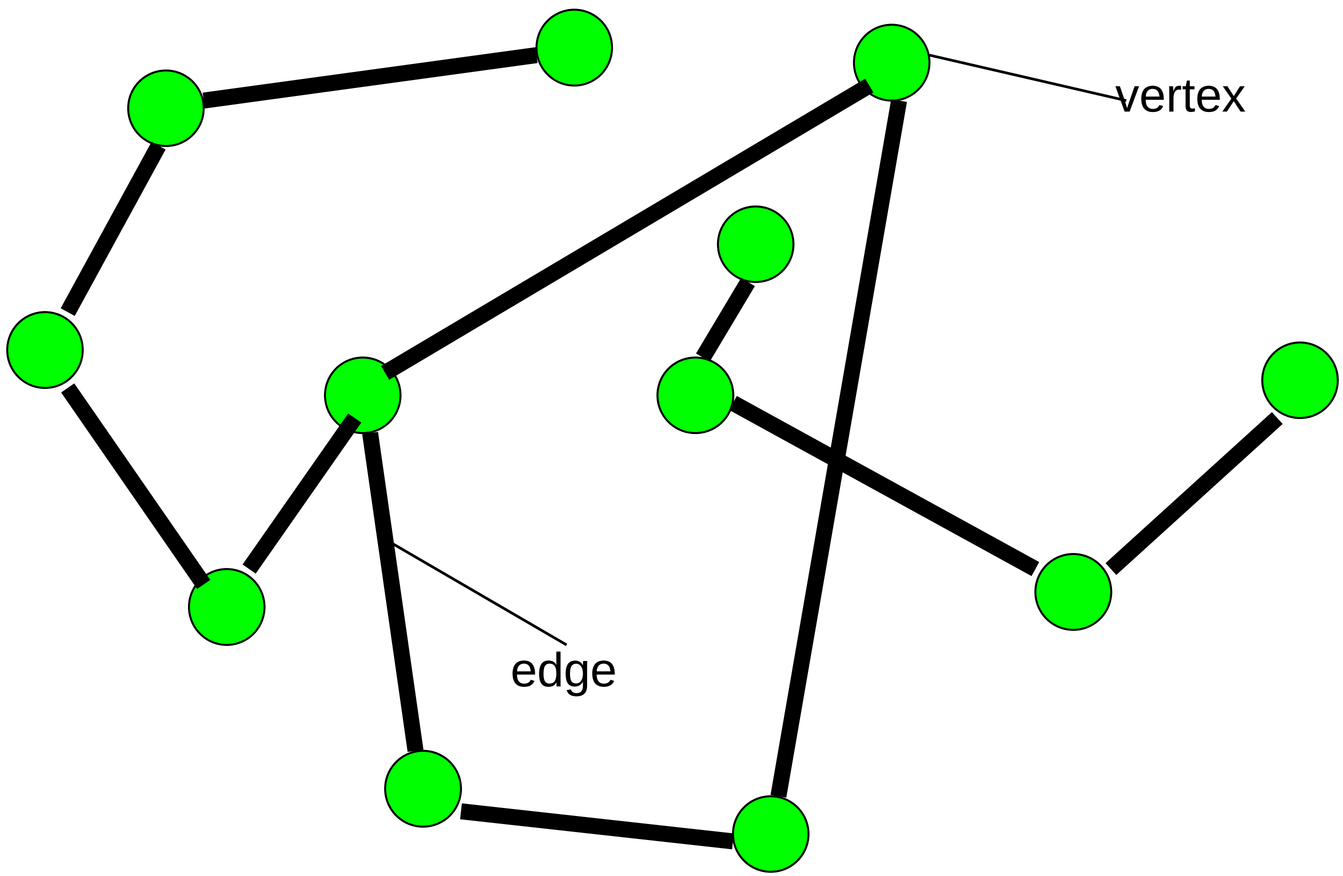
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> v(10);
    vector<int>::iterator I;
    I = v.begin()+5;
    // I points to 6th element
    cout << int(I-v.begin()) << endl;
    // output 5. Iterator is offset by 5 from
    // first element
    system("PAUSE");
}
```

Problem 4

- Write and test a function
  void Replace(list<int>& L)
  which replaces all elements 0 of **L** by 1 (order of elements of **L** should be maintained)
- Note that the function uses pass by reference, so it can indeed change **L**
- For instance, the function should change the list 0-0-1-3-0-5-0-6 to 1-1-3-1-5-1-6.

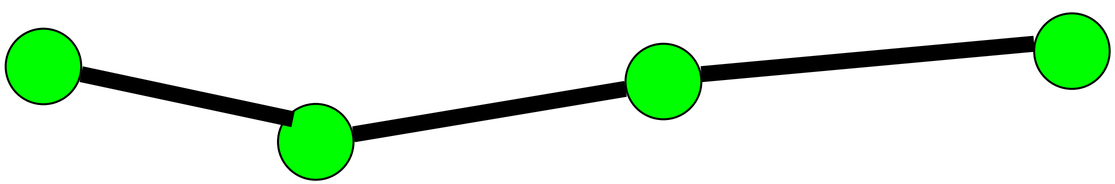
Graphs



Graphs (cont.)

- A graph consists of a finite set of vertices and a set of edges. Each edge is a 2-element set of vertices.
- If {v,w} is an edge, then the vertices v and w are called adjacent

A walk **in a graph is a sequence $v_1, v_2, \ldots, v_r$ of vertices such that any two consecutive vertices are adjacent**

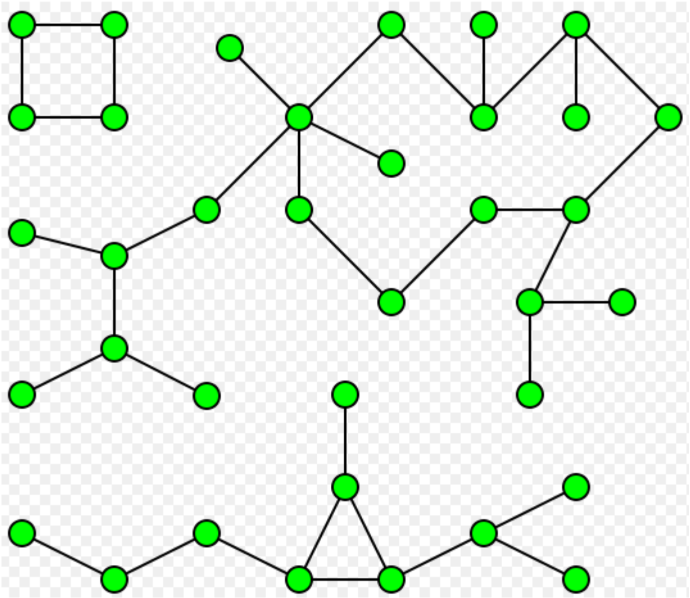


walk

Connected Components

- A set of vertices of a graph is called connected if there is a walk between any two vertices of the set
- A connected component of a graph is a maximal connected set (i.e. it is connected and no point outside the set is adjacent to a point in the set)

Graph with 3 connected components:



Computing Connected Components

Suppose the vertices are 1,…,n

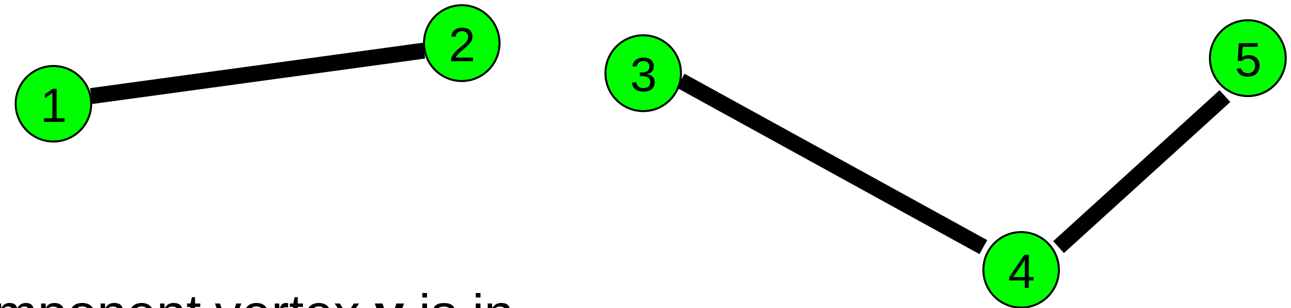
Initialization: C[v]=v and L[v]={v} for v=1,…,n

Procedure: For every edge {a,b} do:

If C[a]≠C[b], then set C[b]=C[a] and L[a] = L[a]∪L[b]

Result: The nonempty final L[v]'s are the connected components

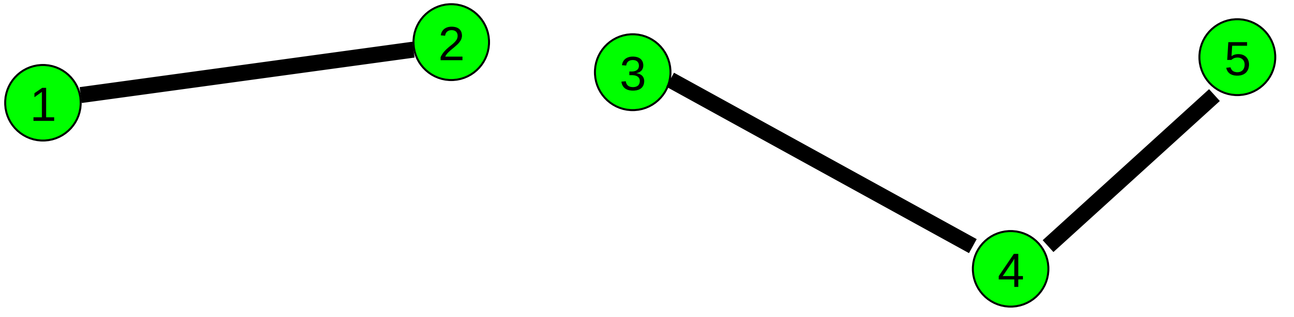
Connected Components: Example



C[v]: component vertex **v** is in
L[v]: list of vertices which are in connected component of **v**
Method: Look at each edge and update **C[v]**, **L[v]** accordingly
Initialization:

v	1	2	3	4	5
C[v]	1	2	3	4	5
L[v]	1	2	3	4	5

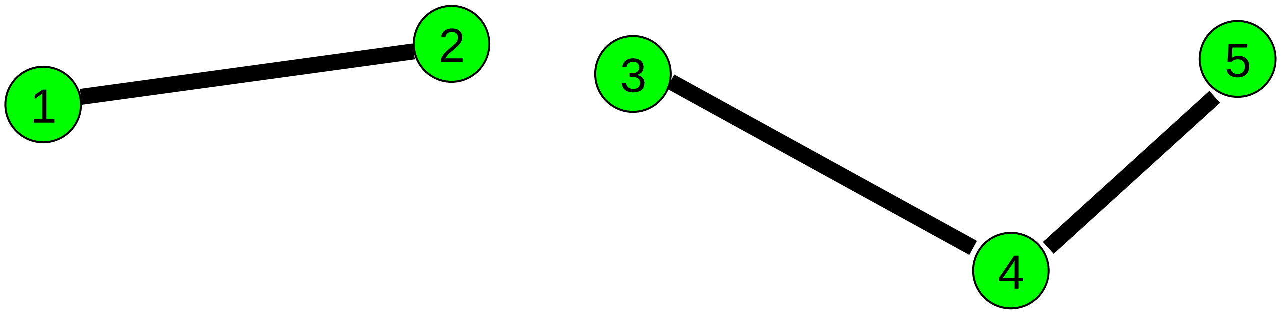
Edge 1-2



Set **C[1]=2**, merge **L[1]** and **L[2]**

v	1	2	3	4	5
C[v]	2	2	3	4	5
L[v]		1,2	3	4	5

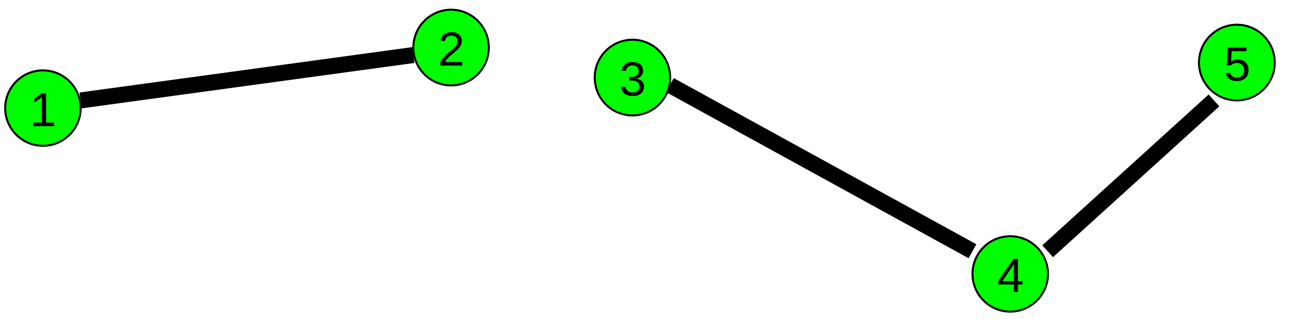
Edge 3-4



Set **C[3]=4**, merge **L[3]** and **L[4]**

v	1	2	3	4	5
C[v]	2	2	4	4	5
L[v]		1,2		3,4	5

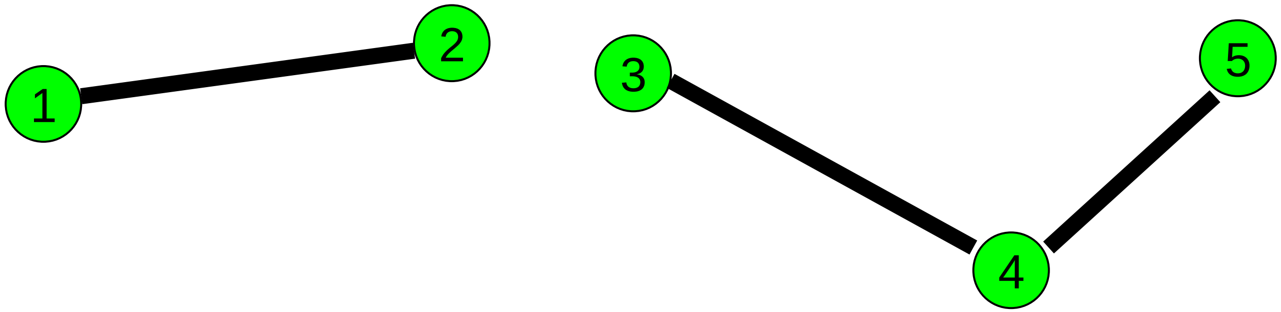
Edge 4-5



Set **C[3]=5**, **C[4]=5**, merge **L[4]** and **L[5]**

v	1	2	3	4	5
C[v]	2	2	5	5	5
L[v]		1,2			3,4,5

Result



v	1	2	3	4	5
C[v]	2	2	5	5	5
L[v]		1,2			3,4,5

Connected components: {1,2}, {3,4,5}

Problem 5

- Download the file graph.txt from NTULearn. It contains the edges of a graph with vertices 1,2,…,20 (one edge in each line)
- Write a program using STL lists that computes the connected components of the graph using the method from the last few slides
- \- Create a vector<list<int> > L of size 21
  \- For i=1,….,20, insert i in L[i]
  \- Create a vector<int> C with C[i]=i for i=1,….,20
  \- For each edge {a,b} of the graph do:
      \- if C[a]==C[b] do nothing. Otherwise:
      \- for all x in L[C[a]] do C[x]=C[b]
      \- Append L[C[a]] to L[C[b]]
- Check if the result is correct.

STL Sets

STL Sets

- To use sets, you need **#include<set>**
- An empty set is created with
  set<type> SetName;
- The type is the type of the elements of the set.
- An element **e** is inserted with **SetName.insert(e);**
- An element **e** is removed with
  SetName.erase(e);
- Printing sets on the screen is completely similar to printing lists
- **SetName.count(e)** returns 1 if **e** is in the set and 0 otherwise

Example

- Create the set {1,2,…,10}
- Delete the element 5
- Print all elements of the set to the screen

```
#include <cstdlib>
#include <iostream>
#include <iterator>
#include <set>
using namespace std;

int main(int argc, char *argv[])
{

    set<int> S;
    for(int i=1;i<=10;i++)
       S.insert(i);
    S.erase(5);

    set<int>::iterator I;
    for(I=S.begin();I!=S.end();I++)
       cout << *I << " ";
    cout << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

//alternatively, use the simplified iteration in C++11

Find Elements in a Set

- Let **S** be an STL set
- If **S** contains **e**, then **S.find(e)** returns an iterator to the postion of **e**
- Otherwise it returns **S.end()**, i.e., the position beyond the last element of **S**
- **find** is the most important function for sets since searching is the most useful application of sets.

Example

```
#include <cstdlib>
#include <iostream>
#include <set>
using namespace std;

int main(int argc, char *argv[])
{
    set<int> S;
    S.insert(2); S.insert(3);
    S.insert(5); S.insert(7);
    for(int n=1;n<=10;n++)
      if(S.find(n)!=S.end())
        cout << n << " is prime " << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Problem 6

Download the files **numbers.txt** and **numbers1.txt** from NTULearn. For each file do:

- Read the numbers contained in the file into a **vector<int> v**
- Find out if **v** contains any number at least twice by comparing **v[i]** with **v[j]** for all pairs (i,j), i<j

Problem 6 (cont.)

For each of the files **numbers.txt, numbers1.txt** do:

- Read the numbers contained in the file into a **set<int> S**
- Use the find function for sets during the construction of S to identify any numbers which occur repeatedly
- How does the speed of this method compare with the method from the previous slide?

STL Algorithms

Overview

- STL provides a number of basic algorithms which operate on containers
- Need **#include<algorithm>**
- Algorithms for searching, sorting, copying, reordering etc.
- See http://www.cplusplus.com/reference/algorithm/ for a list of available algorithms
- We only look at some examples
- Remark: In the following, [a,b) means the range from a to b with a included and b excluded

sort

- Usage: **sort(pos1,pos2)** where **pos1, pos2** are iterator positions
- Effect: sorts the elements in the range [pos1,pos2) in ascending order. Example:

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> v(3);
    v[0]=3; v[1]=2; v[2]=1;
    sort(v.begin(),v.end());
    for(int i=0;i<v.size();i++)
        cout << v[i] << " ";
    system("PAUSE");
}
```

Output:

1 2 3

Problem 7

- Compute the sum of 10 largest numbers in the file numbers.txt from Problem 6
- Method: Read the numbers into a vector, sort it, then compute the sum of last 10 entries of the vector

sort with Criterion

- Usage: **sort(pos1,pos2,criterion)** where **pos1, pos2** are iterator positions and criterion is a function with two parameters of the same type as the container elements and return type bool
- Effect: sorts the elements in the range according to the criterion

```
bool Criterion(int a,int b)
{
    if(a>b)
      return true;
    return false;
}
int main()
{
    vector<int> v(3);
    v[0]=0; v[1]=1; v[2]=2;
    sort(v.begin(),v.end(),Criterion);
    for(int i=0;i<v.size();i++)
        cout << v[i] << " ";
    system("PAUSE");
}
```

Output:

2 1 0

search

- Usage: search(pos1,pos2,pos3,pos4) where pos1,…,pos4 are iterator positions
- Returns the start position of the first occurance of the sequence [pos3,pos4) in [pos1,pos2) if such a subsequence exists
- Returns pos2 otherwise

Example

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> v(10);
    for(int i=0;i<v.size();i++)
        v[i] = i;
    vector<int> sub(2);
    sub[0]=5; sub[1]=6;
    vector<int>::iterator I;
    I= search(v.begin(),v.end(),sub.begin(),sub.end());
    cout << int(I-v.begin()) << endl;
    system("PAUSE");
}
```

Output 5 (first occurrence of 5,6 in v is at index 5)

Problem 8

- Create a random **vector<bool>** of size 1000,000 (use **rand()%2** to create random bools)
- Use the search function to find out if the vector contains a subsequence of 15 consecutive values **true**

C++ Strings

C++ Strings

- string is a class which belongs to the STL
- Purpose: store any sequence of letters, numbers and symbols
- Often used to deal with text
- Declare a C++ string:
  string StringName;
- Needs **#include<string>**
- Assign a sequence of symbols to a string:
  StringName = “sequenceOfSymbols”;

Example

- Create a string **S**
- Assign some text to **S**
- Print **S** to the screen

```
#include <cstdlib>
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string S;
    S="Today is Saturday";
    cout << S << endl;
    system("pause");
}
```

Useful Functions for C++ Strings

- There are lots of functions available for C++ strings
- We only consider some of the most useful functions
- They are most easily understood by example

C++ Strings: Example

```
int main()
{
    string S = "012345";
    S += "yyy";
    string S1="zzz";
    cout << S+S1 << endl;
    cout << S[2] << endl;
    S[2]='a';
    cout << S.size() << endl;
    cout << S.substr(3,2) << endl;
    string s1="abc", s2="abd";
    cout << (s1<s2) << endl;
    system("pause");
}
```

Explanations

- Line 3: a C++ string **S** containing the sequence 01234 of symbols is created
- Line 4: **"yyy"** is appended to **S.** We can use += for appending a string to another string
- Line 5: another string **S1** is created
- Line 6: the concatenation of **S** and **S1** is printed to the screen. We can use + to concatenate strings
- Line 7: the third symbol in **S** is printed to the screen (**S[i]** is the（**i+1**）st symbol in **S**)
- Line 8: the third symbol of **S** is changed to “a”
- Line 9: the number of symbols of **S** is printed to the screen
- Line 10: the substring of **S** starting with fourth symbol and of length 2 is printed to the screen
- Line 11: two new strings are created
- Line 12: the strings are compared lexicographically (according to ascii code). The output will be 1 (means true) since **s1** is lexicographically smaller than **s2**

Problem 9

- Write a program that asks the user to input 5 words (only letters allowed in words).
- The program should print the words to the screen in lexicographic order (with lower/upper case ignored)
- Note: the comparison “<“ for strings does NOT ignore lower/upper case
- Use the sort function with an appropriate criterion
- Useful: tolower(x) converts upper case letters to lowercase letters (**#include<cstdlib>** needed)