


# Introduction to the C++ Standard Library.


For : COP 3330.  
Object oriented Programming (Using C++)  
<http://www.compgeom.com/~piyush/teach/3330>

*Piyush Kumar*




## The C++ Standard Library

- Provides the ability to use:
  - String Types
  - Data Structures (linked list, dynamic arrays, priority queues, binary trees etc)
  - Algorithms (Sorting and Searching...)
  - IO
  - Classes for internationalization support.




## The C++ Standard Library

- Not very homogeneous:
  - String classes are safe and convenient to use (almost self explanatory).
  - The **Standard Template Library (STL)** optimizes for performance and is not required to check for errors. To use it well, you need to understand the concepts and apply them carefully.




## C++ Standard Library

- Containers
  - Objects that hold/contain other objects.
  - Examples: vector, string
- Algorithms
  - Work on the containers
  - Examples: sort, search
- Iterators
  - Provide pointer like interface to containers.



## Other components

- Allocators: Provide memory management for containers. Can be customized.
- Adaptors: A mechanism to make one thing act like another. Example: Stack.
- Function objects: A **function object** or a **functor**, is a construct allowing an object to be invoked or called as if it were an ordinary function,



## Containers

- Of course: Contain objects/built in types.
  - More powerful than arrays.
  - Grow (and shrink?) dynamically
  - Manage their own memory
  - Keep track of their size
  - Allow optimal algorithmic operations like scan, sorts etc.

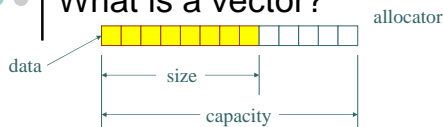
## Containers

- Standard STL sequence Containers:
  - vector, string, deque and list
- Standard non-STL containers:
  - bitset, valarray, priority\_queue, queue.

## Containers

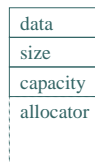
- Prefer sequential containers to arrays.
- Use vector by default
- Use list when there are a lot of insertions/deletions in the middle of the sequence.
- Use deque when there is a lot of insertion at the beginning or the end of the sequence.

## What is a vector?



- A contiguous array of elements
- The first "size" elements are constructed (initialized)
- The last "capacity - size" elements are uninitialized
- Four data members
  - data pointer
  - size
  - capacity
  - allocator

or equivalent



Sample data layout: Internals.

## Vector Interface

```
template <class T, class Allocator = allocator<T> >
class vector {
public:
    ...
    explicit vector(const Allocator& = Allocator());
    explicit vector(size_type n, const T& value = T(),
                  const Allocator& = Allocator());
    ...
    void reserve(size_type n);
    ...
    void resize(size_type sz, const T& c = T());
    ...
    void push_back(const T& x);
    void pop_back();
    ...
    iterator insert(iterator position, const T& x);
    void insert(iterator position, size_type n, const T& x);
    ...
    iterator erase(iterator position);
    iterator erase(iterator first, iterator last);
    ...
    void clear();
};
```

## Vectors

- template <class T, class Allocator = allocator<T> > class vector { ... }
- A default allocator is provided.
- T is the type of the object stored in the vector.
- Constructors for vector:
  - vector<int> ivec1;
  - vector<int> ivec2(3,9);
  - vector<int> ivec3(ivec2);

## Containers : is empty?

- Always use:
  - if(container.empty()) ...
  - Instead of if(container.size() == 0)
- For some containers, calculating size takes linear time.

## An example usage

```
#include <vector>
#include <iostream>

using namespace std;

int main() {
    vector<int> vec(10); // Creates a vector

    // Initializes the vector
    for(int i=0; i< vec.size(); i++) {
        vec[i] = rand() % 10;
        cout << "vec[" << i << "]=" << vec[i] << endl;
    };
    return 0;
}
```

```
pk@linprog4:-->./a.out
vec[0]=3
vec[1]=6
vec[2]=7
vec[3]=5
vec[4]=3
vec[5]=5
vec[6]=6
vec[7]=2
vec[8]=9
vec[9]=1
```

## An example usage

```
#include <vector>
#include <iostream>

using namespace std;

int main() {

    vector<int> ivec;
    cout << ivec[0]; //error
    vector<int> ivec2(10);
    // subscripts available: 0..9
    cout << ivec[10]; // error

    return 0;
}
```

```
pk@linprog4:-->./a.out
Segmentation fault (core dumped)
```

Make this your friend...

## Iterators

- Browsers for containers.
- Allows restricted access to objects stored in a container.
- Can be a class, data structure or an Abstract Data Type.

## Iterators

- A replacement for subscripting, for example in case of vectors: `v[i]`
- Subscripts are not available for all containers but iterators are.
- You can think of an iterator as pointing to an item that is part of a larger container of items.

## Iterators

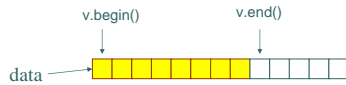
- `Container.begin()` : All containers support a function called `begin`, which will return an iterator pointing to the beginning of the container (the first element)
- `Container.end()` : returns an iterator corresponding to having reached the end of the container. (Not the last element)

## Iterators

- Support the following operations:
  - Operator `*` : Element at the current position (example: `(*it)`). You can use `->` to access object members directly from the iterator. (Like a pointer)
  - Operator `++` : Moves the iterator to the next element. Most iterators will also allow you to use `--` for stepping back one element.
  - Operator `==` and `!=` : Compare two iterators for whether they represent the same position (not the same element).
  - Operator `=` : Assigns an iterator.

## Iterators

- o Vector iterator picture.



- o Reason for half-open range:
  - Easy looping
  - Empty containers → begin() == end()

## Iterators

- o Defining an iterator:

```
std::class_name<template_parameters>::iterator name;
```

- o Example:

```
std::vector<int>::iterator vit = myvec.begin();
cout << (*vit) << endl;
```

- o Printing all elements of a container.

```
std::container_type<template_parameter>::iterator pos;
for ( pos = container.begin();
      pos != container.end(); ++pos)
    cout << (*pos) << endl;
```

## Iterators : Examples

The non-STL way, using subscripts to access data:

```
using namespace std;

vector<int> myIntVector;
// Add some elements to myIntVector
myIntVector.push_back(1); //adds an element to end of vector.
myIntVector.push_back(4);
myIntVector.push_back(8);

for(int y=0; y<myIntVector.size(); y++) {
    cout<<myIntVector[y]<<" "; //Should output 1 4 8
}
```

## Iterators: Examples

```
using namespace std;
vector<int> myIntVector;
vector<int>::iterator myIntVectorIterator;

// Add some elements to myIntVector
myIntVector.push_back(1);
myIntVector.push_back(4);
myIntVector.push_back(8);

for(myIntVectorIterator = myIntVector.begin();
    myIntVectorIterator != myIntVector.end();
    myIntVectorIterator++) {
    cout<<"myIntVectorIterator<<" ";
    // Should output 1 4 8
}
```

## Iterator Types

- o Input Iterator : read only, forward moves.
- o Output Iterator : write only, forward moves.
- o Forward Iterator: Both read/write with (++) support
- o Backward: Both read/write with (--) support
- o Bi-Directional :Read write and Both ++ or -- support.
- o Random: Read/Write/Random access. (Almost act like pointers)

Most Common

## Iterators

Type of iterator	Example
Input Iterator	istream_iterator
Output Iterator	ostream_iterator, inserter, front_inserter, back_inserter
Bi-directional iterator	list, set, multiset, map, multimap
Random access iterator	Vector, deque

## Random Access Iterators

- o Allow arithmetic
  - $it+n$
  - The result will be the element corresponding to the  $n$ th item after the item pointed to be the current iterator.
  - $it - n$  also allowed
  - $(it1 - it2)$  allowed
    - Example: Type of this operation for vectors is defined by `vector<T>::difference_type`.

## Back to vectors

- o Iterator type: Random-access
- o Operator `[]` overloaded

<code>v.size()</code>	Number of elements in vector
<code>v.clear()</code>	Removes all elements
<code>v.pop_back()</code>	Removes last element
<code>v.push_back( elem )</code>	Adds elem at end of vector
<code>v.insert(pos,elem)</code>	Inserts elem at position pos and returns the position of the new element.
<code>v.erase(pos)</code> Another form: <code>v.erase(bpos, epos)</code>	Removes the element at the iterator position pos and returns the position of the next element.

## Back to vectors

<code>v.max_size()</code>	Maximum number of elements possible (in entire memory!).
<code>v.capacity()</code>	Returns maximum number of elements without reallocation
<code>v.reserve(new_size)</code>	Increases capacity to new_size.
<code>v.at( idx )</code>	Returns the element with index idx. Throws range error exception if idx is out of range.
<code>v.front() , v.back()</code>	Returns first , last element.
<code>v.resize(new_size)</code>	Changes the size to new_size.

## Important facts

- o For vectors, the C++ standard states:
  - $\&v[i] = \&v[0] + i$

```
vector< char > vv;
vv.push_back ( 'P' );
vv.push_back ( 'Q' );
vv.push_back ( 'R' );
vv.push_back ( '\0' );
printf("%s\n",&vv[0]);

Output : PQR
```

## The swap trick.

- o To trim capacity, you can use the following trick:
  - `std::vector<T>(v).swap(v);`
  - Makes capacity = size.
    - Example: `vector<int>(ivec2).swap(ivec2);`

## Important Facts

- o When deleting containers of newed/allocated elements, remember to delete/free them before deleting the container.
- o Thread safety of STL containers:
  - Multiple readers are ok
  - Multiple writers to different containers are ok.

## What can a container contain?

- Minimal constraint on elements of a container.
  - Operator=
    - a = b; should be valid
  - A copy constructor
    - YourType b(a); should be valid
- Question :
  - Is vector<int&> allowed?

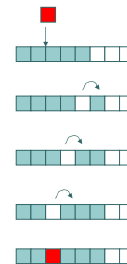
## Suggestions

- Prefer vector and string to dynamically allocated arrays.
- Use reserve() to avoid unnecessary reallocations.
- Avoid using vector<bool>

## Algorithms

- Methods that act on containers (may or may not change them)
  - Examples: Sorting, searching, reversing etc.
  - Examples:
    - sort(v.begin(), v.end())
    - pos = find(v.begin(), v.end(), 3) // returns an iterator in the container.
    - reverse(v.begin(), v.end())
    - unique(v.begin(), v.end()) // operates on a sorted range to collapse duplicate elements.

## Understand Complexity Example: Vector Insert



What happens when the vector is large?

## Intro To The Standard string Class

- C++ has a standard class called "string"
- Strings are simply a sequence of characters
  - Note: This is not a sufficient definition for a "C-string"
  - A "C-string" is an array of characters terminated by a null byte
- Must #include <string> using the standard namespace to get C++ standard string functionality
  - Note: This is different from #include'ing <string.h> which is the header required for "C-string"s
- string variables are used to store names, words, phrases, etc.
- Can be input using ">>" and output using "<<" as other types

## Some string Function

- Declaring a string:
  - string lastName;
  - string firstName("Piyush"); //Note: String literal enclosed in double quotes
  - string fullName;
- Assigning a string:
  - lastName = "Kumar"; //The usual assignment operator
- Appending one string on the end of another:
  - fullName = firstName + lastName; //Results in "PiyushKumar"
  - fullName = firstName + " " + lastName; //Results in "Piyush Kumar"
- Accessing individual characters in a string:
  - myChar = firstName[4]; //Results in 's' (no bounds checking)
  - myChar = firstName.at(4); //Results in 's' (does bounds checking)
- Appending a character to the end of a string:
  - lastName = lastName + myChar; //Results in "Kumars"
- Determining number of characters in string:
  - myInt = firstName.length(); //Results in 6

firstName[5] = 'h'  
firstName[6] is undefined unlike C where its '\0'.

## string Example #1

```
#include <iostream>
#include <string>
using namespace std;
int main(void)
{
    string first;
    string last("Morgan");

    first = "Drew"; //Would be illegal for C-string
    cout << "Length of " << first << " is: " << first.length() << endl;
    cout << "Length of " << last << " is: " << last.length() << endl;

    first += "Morgan";
    cout << "Length of " << first << " is: " << first.length() << endl;
    cout << "Length of " << last << " is: " << last.length() << endl;

    first.assign("Drew");
    first.append(" ");
    first.append(last);
    cout << "Length of " << first << " is: " << first.length() << endl;
    cout << "Length of " << last << " is: " << last.length() << endl;
    return(0);
}
```

```
Length of Drew is: 4
Length of Morgan is: 6
Length of DrewMorgan is: 10
Length of Morgan is: 6
Length of Drew Morgan is: 11
Length of Morgan is: 6
```

## Constructors

```
string() // empty string
string(string s) // copy of s
string(string s, int start) // substring start,end
string(string s, int start, int len) // substring
string(char* a) // copy of C-string
string(int cnt, char c) // one or more chars
string(char* beg, char* end) // [beg, end)
```

## Additional string Functionality

- Strings can be compared with usual operators
  - >, >= (greater than, greater than/equal to)
  - <, <= (less than, less than/equal to)
  - == (equality)
- Strings also have a member function called "compare"
  - int string::compare(string rhs);
  - Return value is negative if calling string is less than rhs
  - Return value is positive if calling string is greater than rhs
  - Return value is zero if both strings are identical

## Other overloaded operators

- = is used to assign a value (char, C-string, or string) to a string.
- += is used to append a string, character, or C-string to a string.
- + is used to concatenate two strings or a string with something else
- << and >> are used for input and output. On input, leading whitespace is skipped, and the input terminates with whitespace or end of file.

## When you need a C-string

```
string s = "1234";
s.data() // returns s as a data array, no '\0'.
s.c_str() // returns s as a C-string with '\0'
int i = atoi(s.c_str()); // conversion
// i is now 1234.
```

```
char *carray = new char[80];
s.copy(carray, 79); // copies up to 79 char
```

## String Operations

```
s.append(s2); // append s2 to s
s.push_back(c); // append a char
s.erase( various ); // erases substrings
s.insert( various ); // inserts substrings
s.clear(); // removes all contents
s.resize(cnt); // change the size of s to cnt
swap(a, b); // for general containers.
```

## String Operations

`s.replace(whatever);` // replaces characters  
`s.size();` or `s.length();` // how many characters?  
`s.max_size();` // maximum number of char?  
`s.empty();` // is s empty?  
`s.reserve(cnt);` // reserves memory

## string Example #2

```

int main(void)
{
    string s1 = "Drew";
    string s3;
    int result;

    s3 = "Bob";
    if (s3 < s1)
        cout << "oper: s3 less than s1";
    if (s3 > s1)
        cout << "oper: s3 greater than s1";
    if (s3 == s1)
        cout << "oper: s3 is equal to s1";
    cout << endl;

    result = s3.compare(s1);
    if (result < 0)
        cout << "comp: s3 less than s1";
    else if (result < 0)
        cout << "comp: s3 greater than s1";
    else
        cout << "comp: s3 is equal to s1";
    cout << endl;
}
  
```

```

oper: s3 less than s1
comp: s3 less than s1
oper: s3 is equal to s1
comp: s3 is equal to s1
  
```

## Even More string Functionality

- Getting a substring of a string:
  - `string string::substr(int startPos, int length)`
    - Returns the substring starting at "startPos" with length of "length"
- Finding the location of a substring within a string:
  - `int string::find(string lookFor);`
    - Returns the index where the first instance of "lookFor" was found in the string
    - Returns "string::npos" (which is usually -1) when the substring isn't found
  - `int string::find(string lookFor, int startFrom);`
    - Returns the index where the first instance of "lookFor" was found, starting the search at the index "startFrom", or "string::npos" when the substring isn't found
- Finding specific characters in a string:
  - `int string::find_first_of(string charList, int startFrom);`
    - Returns the index of the first instance of any character in "charList", starting the search at the index "startFrom", or "string::npos" if none of the chars are found
  - `int string::find_first_not_of(string charList, int startFrom);`
    - Returns the index of the first instance of any character NOT in "charList", starting the search at the index "startFrom", or "string::npos" if none of the chars are found

## string Example #3

```

int main()
{
    int startPos;
    int len;
    int commaLoc;
    int howLoc;
    int spaceLoc;
    string myStr;
    string myStr2;

    myStr = "Hello, how are you?";
    startPos = 7;
    len = 3;
    myStr2 = myStr.substr(startPos, len);
    cout << "Substr: " << myStr2 << endl;
    commaLoc = myStr.find(",");
    howLoc = myStr.find(myStr2);
    cout << "Comma: " << commaLoc;
    cout << " how: " << howLoc << endl;

    cout << "Spaces:";
    spaceLoc = myStr.find(" ");
    while (spaceLoc != string::npos)
    {
        cout << " " << spaceLoc;
        spaceLoc = myStr.find(" ", spaceLoc + 1);
    }
    cout << endl;

    cout << "Punct and spaces:";
    loc = myStr.find_first_of(" ,? ", 0);
    while (loc != string::npos)
    {
        cout << " " << loc;
        loc = myStr.find_first_of(" ,? ", loc + 1);
    }
    cout << endl;
    return (0);
}
  
```

```

Substr: how
Comma: 5 how: 7
Spaces: 6 10 14
Punct and spaces: 5 6 10 14 18
  
```

## string Class Implementation

- The string class uses dynamic memory allocation to be sure segmentation faults don't occur
  - When a string is updated such that it requires more characters than currently allocated, a new, larger array is allocated and the prior contents are copied over as necessary
- Since dynamic allocation is relatively slow, it is not desirable to be re-allocating strings often
  - C++ allows some memory to be "wasted" by often allocating more space than is really needed
  - However, as strings are appended to the end, it is likely that a re-allocation won't be needed every time
  - Occasionally, re-allocation is necessary and is performed, again allocating more memory than necessary
- Note: this is all done *automatically* by the string class ( Similar to vectors? )

## Some Final string Functionality

- Several member functions are available to get information about a string
  - `capacity`: The number of characters that can be placed in a string without the inefficiency of re-allocating
  - `length`: The number of characters currently in the string
- You can manually change the capacity of a string
  - `resize`: Sets the capacity of a string to be at least a user-defined size
  - This can be useful if you know a string will be at most *n* characters long
    - By resizing the string to capacity *n* only that amount of memory is associated with the string
    - This prevents wasted memory when you know the exact size you need
    - Additionally, it can help prevent numerous re-allocations if you will be appending on to the end of the string, but know the final size ahead of time

### Example #4

```

int main(void)
{
    string str1;
    string str2;

    cout << "Str: " << str << endl;
    cout << "Length: " << str.length();
    cout << " Cap: " << str.capacity();
    cout << endl;

    str = "888";
    cout << "Str: " << str << endl;
    cout << "Length: " << str.length();
    cout << " Cap: " << str.capacity();
    cout << endl;

    str += "-111-";
    cout << "Str: " << str << endl;
    cout << "Length: " << str.length();
    cout << " Cap: " << str.capacity();
    cout << endl;

    str += "1723-9";
    cout << "Str: " << str << endl;
    cout << "Length: " << str.length();
    cout << " Cap: " << str.capacity();
    cout << endl;

    str += "abcdefghijklmnopqrstuv";
    cout << "Str: " << str << endl;
    cout << "Length: " << str.length();
    cout << " Cap: " << str.capacity();
    cout << endl;

    return (0);
}

```

```

Str:
Length: 0 Cap: 0
Str: 888
Length: 3 Cap: 31
Str: 888-111-
Length: 8 Cap: 31
Str: 888-111-1723-9
Length: 14 Cap: 31
Str: 888-111-1723-9abcdefghijklmnopqrstuv
Length: 36 Cap: 63

```

### C Vs C++: Strings

C Library Functions	C++ string operators /member functions.
strcpy	=
strcat	+=
strcmp	=, !=, <, >, <=, >=
strchr, strstr	.find( ) method
strrchr	.rfind( ) method
strlen	.size( ) or .length( ) methods

### Reading text into a string

getline(istream, s); // Reads from istream (e.g., cin or a file) into the string s. Returns a reference to istream that can be used again.

- Reads all characters until a line delimiter or end of file is encountered.
- The line delimiter is extracted but not put into the string
- You can then parse s without worrying about end of line or end of file characters.

### Char functions in C/C++

- #include <ctype.h>
- int isalnum(int c); //non-zero iff c is alphanumeric
- int isalpha(int c); //non-zero iff c is alphabetic
- int isdigit(int c); //non-zero iff c is digit: 0 to 9
- int islower(int c); //non-zero iff c is lower case
- int ispunct(int c); //non-zero iff c is punctuation
- int isspace(int c); //non-zero iff c is a space char
- int isupper(int c); // non-zero iff c is upper case
- int isxdigit(int c); //non-zero iff c is hexadecimal
- int tolower(int c); //returns c in lower case
- int toupper(int c); //returns c in upper case

### Using the transform algorithm

#include <algorithm>

- Lowercase all characters of a string:
 

```
transform(s.begin(), s.end(), // source
          s.begin(), // destination
          tolower); // operation
```
- Uppercase all characters:
 

```
transform(s.begin(), s.end(), s.begin(), toupper);
```
- tolower and toupper are C-string functions. Other functions can also be used.

### Using the transform algorithm

#include <algorithm>

- What does the following do?
  - If (s == reverse(s.begin(),s.end()))
 

```
cout << "S is a ...";
```

## Solution to assignment 1

```
vector<string> vs_inpvec;
string s_one_entry;
int counter = 0;
while( getline( cin, s_one_entry) ){
    int _pos = s_one_entry.rfind('/');
    if(_pos != -1)
        s_one_entry = s_one_entry.substr(_pos+1, s_one_entry.size());

    vs_inpvec.push_back(s_one_entry);
}
```

What does this code do?

## Solution to assignment 1

```
sort(vs_inpvec.begin(), vs_inpvec.end());
int i_inp_elements = vs_inpvec.size();
cout << "Input Elements   = " << i_inp_elements << endl;

vs_inpvec.erase(
    unique(vs_inpvec.begin(), vs_inpvec.end()) ,
    vs_inpvec.end()
);

int i_unique_elements = vs_inpvec.size();
cout << "Unique Elements   = " << i_unique_elements << endl;
cout << "Duplicate Elements = " <<
    i_inp_elements - i_unique_elements
<< endl;
```

Reading assignment: Chapter 3, 9  
Including Bitset.

## #include <bitset>

```
00000000
10000011
```

- o Ordered collection of bits.
- o Example program

```
// create a bitset that is 8 bits long
bitset<8> bs;
// display that bitset
for( int i = (int) bs.size()-1; i >= 0; i-- ) { cout << bs[i] << " "; }
cout << endl;

// create a bitset out of a number bitset<8>
bs2( (long) 131 ); // display that bitset, too
for( int i = (int) bs2.size()-1; i >= 0; i-- ) { cout << bs2[i] << " "; }
cout << endl;
```

## bitset operators

- o != returns true if the two bitsets are not equal.
- o == returns true if the two bitsets are equal.
- o &= performs the AND operation on the two bitsets.
- o ^= performs the XOR operation on the two bitsets.
- o |= performs the OR operation on the two bitsets.
- o ~ reverses the bitset (same as calling flip())
- o <<= shifts the bitset to the left
- o >>= shifts the bitset to the right
- o [x] returns a reference to the xth bit in the bitset.

## Example

```
// create a bitset out of a number
bitset<8> bs2( (long) 131 );
cout << "bs2 is " << bs2 << endl;

// shift the bitset to the left by 4 digits
bs2 <<= 4;
cout << "now bs2 is " << bs2 << endl;
```

Output?  
bs2 is 10000011  
now bs2 is 00110000

Recommended Exercise:  
3.23, 3.21, 3.18, 3.15