# Strings

# The Standard C++ string Class

Standard C++ includes a new class called string. This class improves on the traditional Cstring in many ways. For one thing, you no longer need to worry about creating an array of the right size to hold string variables. The string class assumes all the responsibility for memory management. Also, the string class allows the use of overloaded operators, so you can concatenate string objects with the + operator:

s3 = s1 + s2

There are other benefits as well. This new class is more efficient and safer to use than C-strings were. In most situations it is the preferred approach. (However, as we noted earlier, there are still many situations in which C-strings must be used.) In this section we'll examine the string class and its various member functions and operators.

# Defining and Assigning string Objects

```
#include <string>

string s1(“Man”);                    //initialize
string s2 = “Beast”;                 //initialize
string s3;
s3 = s1;                             //assign
cout << “s3 = “ << s3 << endl;
s3 = “Neither “ + s1 + “ nor “;      //concatenate
s3 += s2;                            //concatenate
cout << “s3 = “ << s3 << endl;
s1.swap(s2);                         //swap s1 and s2
cout << s1 << “ nor “ << s2 << “\n”;
```

# Input/Output with string Objects

```
int main() {                                                    //objects of string class
        string full_name, nickname, address;
        string greeting("Hello, ");
        getline(cin, full_name);                                //reads embedded blanks
        cout << "Your full name is: " << full_name << endl;
        cin >> nickname;                                        //input to string object
        greeting += nickname;                                   //append name to greeting
        cout << greeting << endl;                               //output: "Hello, Jim"
        cout << "Enter your address on separate lines\n";
        cout << "Terminate with '$'\n";
        getline(cin, address, '$');                             //reads multiple lines
        cout << "Your address is: " << address << endl;
        return 0;
}
```

# Finding string Objects

```
int main() {
        string s1 = “In Xanadu did Kubla Kahn a stately pleasure dome decree”;
        int n;       n = s1.find(“Kubla”);
        cout << “Found Kubla at “ << n << endl;
        n = s1.find_first_of(“spde”);
        cout << “First of spde at “ << n << endl;
        n = s1.find_first_not_of(“aeiouAEIOU”);
        cout << “First consonant at “ << n << endl;
        return 0;
}
```

# Modifying string Objects

```
int main() {
    string s1(“Quick! Send for Count Graystone.”);
    string s2(“Lord”);
    string s3(“Don’t “);
    s1.erase(0, 7);              //remove “Quick! “
    s1.replace(9, 5, s2);        //replace “Count” with “Lord”
    s1.replace(0, 1, “s”);       //replace ‘S’ with ‘s’
    s1.insert(0, s3);            //insert “Don’t “ at beginning
```

# Modifying string Objects

```
    s1.erase(s1.size()-1, 1);                    //remove '.'
    s1.append(3, '!');                           //append "!!!"
    int x = s1.find(' ');                        //find a space
    while( x != -1 ) {                       //loop while spaces remain
        s1.replace(x, 1, "/");                   //replace with slash
        x = s1.find(' ');                        //find next space
    }
    cout << "s1: " << s1 << endl;
    return 0;
}
```

# Comparing string Objects

```
int main()
{
        string aName = “George”;
        string userName;
        cout << “Enter your first name: “;
        cin >> userName;
        if(userName==aName)                          //operator ==
                cout << “Greetings, George\n”;
        else if(userName < aName)                    //operator <
                cout << “You come before George\n”;
```

# Comparing string Objects

```
    else cout << “You come after George\n”;
                                                    //compare() function
    int n = userName.compare(0, 2, aName, 0, 2);
    cout << “The first two letters of your name “;
    if(n==0) cout << “match “;
    else if(n < 0) cout << “come before “;
    else cout << “come after “;
    cout << aName.substr(0, 2) << endl;
    return 0;
}
```

# Accessing Characters in string Objects

```
char charray[80];        string word;        cin >> word;
int wlen = word.length();                    //length of string object
cout << “One character at a time: “;
for(int j=0; j<wlen; j++)
      cout << word.at(j);                    //exception if out-of-bounds
      // cout << word[j]; //no warning if out-of-bounds
word.copy(charray, wlen, 0);                 //copy string object to array
charray[wlen] = 0;                           //terminate with ‘\0’
cout << “\nArray contains: “ << charray << endl;
```

# String Member Functions

```
cin >> str                          // takes the string without spaces, ‘space‘ terminator.
getline(cin, str)                       // takes the whole string with spaces, ‘\n’ terminator.
str.size()                          // returns length of string
str = “Hello”                       // replace with a new string
str.clear()                         // clear string
str.empty()                         // test if string is empty
str[index]                          // get character of string
str += “Hello”                      // append new string at the end
str.insert(pos, “newString”)        // insert at pos
str.erase(pos, count)               // erase count char starting from pos
str.substr(pos, count)              // generate a string starting at pos with length count.
```

# References and Homework

- Object-Oriented Programming in C++ (4th Edition) - Chapter 7
- http://www.cplusplus.com/reference/string/string/string/

# STL 1

# Introduction to the STL

The STL contains several kinds of entities. The three most important are containers, algorithms, and iterators.

A container is a way that stored data is organized in memory.

Algorithms in the STL are procedures that are applied to containers to process their data in various ways.

Iterators are a generalization of the concept of pointers: they point to elements in a container.

# Containers

A container is a way to store data, whether the data consists of built-in types such as int and float, or of class objects. The STL makes **seven basic kinds of containers available**, as well as three more that are derived from the basic kinds.
In addition, **you can create your own containers based on the basic kinds**. You may wonder why we need so many kinds of containers. Why not use C++ arrays in all data storage situations? The answer is  efficiency. An array is awkward or slow in many situations.

# Pairs, first, second, make_pair

```
#include <utility>     // std::pair, std::make_pair
#include <string>      // std::string
#include <iostream>    // std::cout
using namespace std;

int main () {
  pair <string,double> product1;                  // default constructor
  pair <string,double> product2 ("tomatoes",2.30);   // value init
  pair <string,double> product3 (product2);         // copy constructor

  product1 = make_pair("lightbulbs",0.99);   // using make_pair (move)

  product2.first = "shoes";                 // the type of first is string
  product2.second = 39.90;                    // the type of second is double

  cout << "The price of " << product1.first << " is $" << product1.second << '\n';
  cout << "The price of " << product2.first << " is $" << product2.second << '\n';
  cout << "The price of " << product3.first << " is $" << product3.second << '\n';
  return 0;
}
```

# Sequence Containers

A sequence container stores a set of elements in what you can visualize as a line, like houses on a street. Each element is related to the other elements by its position along the line. Each element (except at the ends) is preceded by one specific element and followed by another.
An ordinary C++ array is an example of a sequence container.

| Container | Characteristic | Advantages and Disadvantages |
| --- | --- | --- |
| ordinary C++ array | Fixed size | • Quick random access (by index number)<br>• Slow to insert or erase in the middle Size cannot be changed at runtime |
| vector | Relocating, expandable array | • Quick random access (by index number)<br>• Slow to insert or erase in the middle Quick to insert or erase at end |
| deque | Like vector, but can be accessed at either end | • Quick random access (using index number)<br>• Slow to insert or erase in the middle<br>• Quick insert or erase (push and pop) at either the beginning or the end |

# Member Functions

Some Member Functions Common to All Containers

| Name | Purpose |
|---|---|
| size() | Returns the number of items in the container |
| empty() | Returns true if container is empty |
| begin() | Returns an iterator to the start of the container, for iterating forwards through the container |
| end() | Returns an iterator to the past-the-end location in the container, used to end forward iteration |
| rbegin() | Returns a reverse_iterator to the end of the container, for iterating backward through the container |
| rend() | Returns a reverse_iterator to the beginning of the container; used to end backward iteration |

# Container Adapters

It’s possible to create special-purpose containers from the normal containers mentioned previously using a construct called container adapters.

| Container | Implementation | Characteristics |
| --- | --- | --- |
| stack | Can be implemented as vector, or deque | Insert (push) and remove (pop) at one end only |
| queue | Can be implemented as deque | Insert (push) at one end, remove (pop) at other |
| priority queue | Can be implemented as vector or deque (low efficiency) | Insert (push) in random order at one end, remove (pop) in sorted order from other end |

# Vectors - push_back(), size(), and operator[]

```
#include <vector>
int main() {
	vector<int> v;		//create a vector of ints
	v.push_back(10);		//put values at end of array
	v.push_back(11);
	v.push_back(12);
	v.push_back(13);
	v[0] = 20;			//replace with new values
	v[3] = 23;
	for(int j=0; j<v.size(); j++)	//display vector contents
		cout << v[j] << ' ';	//20 11 12 23
	cout << endl;
	return 0;
}
```

# Vectors - swap(), empty(), back(), and pop_back()

```
#include <vector>
int main() {                                        //an array of doubles
        double arr[] = { 1.1, 2.2, 3.3, 4.4 };
        vector<double> v1(arr, arr+4);              //initialize vector to array
        vector<double> v2(4);                       //empty vector of size 4
        v1.swap(v2);                                //swap contents of v1 and v2
        while( !v2.empty() ){                   //until vector is empty,
                cout << v2.back() << ' ';           //display the last element
                v2.pop_back();                      //remove the last element
        }                                        //output: 4.4 3.3 2.2 1.1
        cout << endl;
        return 0;
}
```

# Vectors - insert() and erase()

```
int main() {
        int arr[] = { 100, 110, 120, 130 };                     //an array of ints
        vector<int> v(arr, arr+4);                              //initialize vector to array
        cout << "\nBefore insertion: ";
        for(int j=0; j<v.size(); j++)     cout << v[j] << ' ';
        v.insert( v.begin()+2, 115);                            //insert 115 at element 2
        cout << "\nAfter insertion: ";
        for(j=0; j<v.size(); j++)         cout << v[j] << ' ';
        v.erase( v.begin()+2 );                                 //erase element 2
        cout << "\nAfter erasure: ";
        for(j=0; j<v.size(); j++)         cout << v[j] << ' ';
        cout << endl;
        return 0;
}
```

# Deques - push_back(), push_front(), front()

```
#include <deque>
int main() {
	deque<int> deq;
	deq.push_back(30);		//push items on back
	deq.push_back(40);
	deq.push_back(50);
	deq.push_front(20);		//push items on front
	deq.push_front(10);
	deq[2] = 33;				//change middle item
	for(int j=0; j<deq.size(); j++)
		cout << deq[j] << ' ';		//display items
	cout << endl;
	return 0;
}
```

# Stack

Last in first out == deque with operations push_front() and pop_front()

# queue

First in first out == deque with operations push_back() and pop_front()

Questions ?

**Thank You :)**