MH1402 Algorithms & Computing II

Lecture 8 Strings, Searching & Sorting

Wu Hongjun

Overview

- **C++ Strings**
  - **Declaration**
  - **Comparison**
  - **Member functions**
- **Bubble Sort**
- **Binary Search**

C++ Strings

- **A C++ string is a class that holds a sequence of characters**
  - **A C++ string is a vector of chars**
  - **C++ string is significantly different from C string**
    - **C string is an array of chars ended by a byte with value zero**
    - **We do not learn C string in this course**

C++ Strings

- **The declaration of a C++ string is the same as that for variables**
  - **Always use #include <string> to make the code portable**
  - **#include<string> is not needed in Mingw-gcc in CodeBlocks, but other compilers may require it**

Example:

```
#include <string>
using namespace std;
int main()
{
    // …….
    string  foo;                          // declare an empty string with name foo
    string  bar = "Hello, World!";  // declare and initialize a string bar
```

C++ Strings

```
string foo;                    // declare a string with name foo
foo = "Hello, World!";    // initialize foo
//The 1st  element of foo is foo[0],  its value is character ‘ H ’;
//The 2nd element of foo is foo[1],  its value is character ‘ e ’;
//The 3rd element of foo is foo[2],  its value is character ‘ l ’;
//The 4th element of foo is foo[3],  its value is character ‘ l ’;
//The 5th element of foo is foo[4],  its value is character ‘ o ’;
//The 6th element of foo is foo[5],  its value is character ‘ , ’;
//The 7th element of foo is foo[6],  its value is character ‘ ’(space);
//……
//The last element   of foo is foo[12],its value is character ‘!’;
```

C++ Strings

- **Concatenation of strings with plus sign +**

```
string surname, firstname, fullname;
surname   = "Foo";
firstname = "Bar";

fullname  = firstname + surname;
cout << fullname << endl;        //print: BarFoo

fullname  = firstname + " " + surname;
cout << fullname << endl;        //print: Bar Foo
```

C++ Strings Member Functions

- C++ string is a vector with character elements
- There are many string member functions

  http://www.cplusplus.com/reference/string/string/
- C++ string can use the vector member functions:
  - at, push_back, pop_back, clear, resize, size, begin, end, erase, insert
- Some string member functions are not available for vector
  - length, find, substr, compare, append ….

pop_back of C++ string is supported in the latest C++11
Not supported in the current Code::Blocks default compiler

C++ Strings Member Functions

- **Member function length( )**

  Example: string s = "Bar Foo";
  cout << s.length( ); // print 7

  - **It is equivalent to the member function size( )**

    cout << s.size( ); // print 7

  - **Note that <u>vector does not have the length( ) member function</u>**

C++ Strings Member Functions

- **Member function substr( ) extracts part of a string**

  string s.substr(int start, int len)

  extracts *len* elements from string s, starting from the position *start*,

  return the extracted string;

  substr() example 1:

  ```
  string s = "Bar Foo";
  string s1 = s.substr(2, 4);       // s1 = "r Fo"
  // the first number in substr( ) is the index of the starting element;
  // the second number is the length of the extracted string.
  ```

C++ Strings Member Functions

substr() example 2:

```
string s = "Bar Foo";
string s1 = s.substr(2, 9);     // s1 = "r Foo"
          // the extracted string does not exceed the bound of the string.
```

C++ Strings Member Functions

substr() example 3:

```
string s = "Bar Foo";
string s1 = s.substr(7, 9);   // s1 is an empty string  since 7 is the length of s
```

substr() example 4:

```
string s = "Bar Foo";
string s1 = s.substr(8, 9);  // program crashes since 8 is larger than the length of s
```

C++ Strings Member Functions

- **Member function find( )**

  int s.find(string s2, int start)

  Starting search from position "start" of string s,

  returns starting position of string s2 in string s.

C++ Strings Member Functions

find( ) example 1:

```
//                  012345678901234567
string foo = "This is interesting";
```

cout << foo.find("is", 0) << endl; // print 2 "This is interesting"
cout << foo.find("is", 3) << endl; // print 5 "This is interesting"
cout << foo.find("in", 0) << endl; // print 8 "This is interesting"
cout << foo.find("in", 9) << endl; // print 16 "This is interesting"

C++ Strings Member Functions

find( ) example 2:

```
string foo = "This is interesting";
cout << foo.find(“tho", 0) << endl;
        // print 4294967295  (largest unsigned int)
        //  it indicates “no match”
```

C++ Strings Member Functions

find( ) example 3:
The following code is preferred when using find()

```
string foo = "This is interesting";
unsigned long long found = foo.find("is", 0);
if (found != string::npos)   //string::npos is the largest unsigned int
{
   cout << "The subsring is found at the position: " << found << endl;
}
```

C++ Strings Member Functions

- **Member function insert( )**

  insert() is overloaded with many functions, we learn the following:

s.insert(iterator p, char c)
// similar to insert() of vector
// inserts a single character c at the iterator p.

s.insert(int start, string s1)
// different from insert in vector
// inserts string s1 into s at starting position start

C++ Strings Member Functions

insert() example 1 (using iterator):

```
string s = "Bar Foo";
s.insert(s.begin()+2, 'z');     // s becomes "Bazr Foo"
```

insert() example 2:

```
string s = "Bar Foo";
s.insert(s.begin()+7, 'z');  // s becomes "Bar Fooz"
```

insert() example 3:

```
string s = "Bar Foo";
s.insert(s.begin()+8, 'z');  // program crashes
```

C++ Strings Member Functions

insert() example 4 :

```
string s = "Bar Foo";
string s1 = "def";
s.insert(2, s1);                // s becomes "Badefr Foo"
```

insert() example 5:

```
string s = "Bar Foo";  string s1 = "def";
s.insert(7, s1);                // s becomes "Bar Foodef"
```

insert() example 3:

```
string s = "Bar Foo";  string s1 = "def";
s.insert(8, s1);                // program crashes
```

C++ String Comparison

- **Character comparison is the same as comparing the integer values of the characters (ASCII table) : space < '0' < '1' < … < '9' < 'A' < 'B' < … < 'Z' < 'a' < 'b' <… < 'z'**

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

C++ String Comparison

- **String comparison:**
  - **For two strings foo and bar, we compare them starting from the first elements. If foo[0] is smaller than bar[0], then foo is smaller than bar.**
  - **If the first k characters of foo and bar are the same, but foo[k] and bar[k] are different, then comparing bar and foo is the same as comparing foo[k] and bar[k]**
  - **For example:**

```
string foo = "apple", bar = "orange", qux = "Orange";
foo > bar      returns false
foo > qux      returns true
bar > qux      returns true
```

C++ String Comparison

- **String comparison (cont.)**
  - **If the length of foo and bar are different, foo.length() > bar.length(), and the first bar.length() elements of foo and bar are the same, then foo is larger than bar.**

Example: string foo = "apple", bar = "app";

foo > bar returns true
foo == bar returns false
foo == foo returns true

C++ String Comparison

- **Alternatively, string member function compare() can be used to compare two strings**

int s1.compare(string s2)

The function returns a negative integer if s1 < s2

0 if s1 = s2

a positive integer if s1 > s2

Example:

```
string foo, bar;
foo = "apple"; bar = "orange";
cout << foo.compare(bar);   // print a negative number
cout << bar.compare(foo);   // print a positive number
```

C++ Strings: getline

- **How to get string from keyboard?**

  Example: string foo, bar, qux;

  cin >> foo >> bar >> qux;

- **We encounter a problem: what would happen if there are spaces in the input string?**
  - **Note that when we use "cin >>" , a space indicates the end of an input (here a string)**
  - **The solution is to use getline() when the input string contains space(s)**

C++ Strings : getline

- **We can use the getline function to read all characters, including spaces, into a string until ENTER is pressed**

Example:

```
string foo;
getline(cin, foo);      /*whatever you typed would be stored into foo,
                                until the newline character '\n' (ENTER) */
cout << foo << endl;
```

C++ Strings : getline

- **If getline follows a cin (and the cin is finished by ENTER), there will be some problem:**

```
string foo, bar;
cin >> bar;              // suppose we finish it by pressing ENTER ;
getline(cin, foo);
/* the user cannot input data into foo. The reason is that getline
   gets an new line character '\n' (ENTER) from the previous cin, then
   consider the input finishes */
```

C++ Strings : getline

- **To solve the problem given in the previous slide, if both cin and getline are used to get data from user in your code, you use cin.ignore(1000,'\n') immediately after every cin (to remove the character '\n' from cin)**
  Example:

```
cin << foo1;
cin.ignore(1000, '\n');
//some codes …
getline(cin,qux);

//some codes …

cin << foo2;
cin.ignore(1000,'\n');
```

cin.ignore(1000,'\n') means that either next 1000 characters or the characters until '\n' shall be ignored, whichever comes first.

C++ Strings : getline

- **Note that when we use cin >> , the spaces are automatically ignored**

```
string foo;
cin >> foo;     // suppose the user enter the string "   abcdef  "
                // the string foo becomes "abcdef"
```

- **But getline does not ignore the spaces:**

```
string foo;
getline(cin, foo);  // suppose the user enter the string "   abcdef  "
                                // the string foo becomes "   abcdef  "
 cout << foo.size();   // print 11 (3 spaces before a, 2 spaces after f)
```

Searching and Sorting

Searching & Sorting

- **Searching and Sorting are two types of widely used algorithms**
  - **Simple examples:**

    find a word in a document

    sort the dictionary for convenient lookup

    sort the phonebook (by names or numbers) for convenient lookup
  - **Very complicated example:**

    Google search engine finds the web pages according to the keywords you typed (involving a lot of searching and sorting).
- **We have already practiced some simple and important searching algorithm in Lab 5:**
  - **Find the minimum (maximum) in an array**

Searching & Sorting

- **There are many searching/sorting algorithms**

  http://en.wikipedia.org/wiki/Sorting_algorithm

  http://en.wikipedia.org/wiki/Search_algorithm

- **Today we learn only one sorting algorithm and one searching algorithm**
  - **Bubble sort**
  - **Binary search (find elements in a sorted array/vector)**

Bubble sort

- **Bubble sort**
  - **We keep moving the small elements in one direction of an array/vector**
  - **It is like the bubble in liquid: the bubble (light) moving up gradually**

Bubble sort

- **Suppose we have an array A with n elements,**
  sort from smallest to largest

Step 0. Compare A[0] with A[1], if A[0] is larger than A[1], swap A[0] and A[1];
Step 1. Compare A[1] with A[2], if A[1] is larger than A[2], swap A[1] and A[2];
Step 2. Compare A[2] with A[3], if A[2] is larger than A[3], swap A[2] and A[3];
……..
Step n-2. Compare A[n-2] with A[n-1], if A[n-2] larger than A[n-1],
swap A[n-2] and A[n-1];

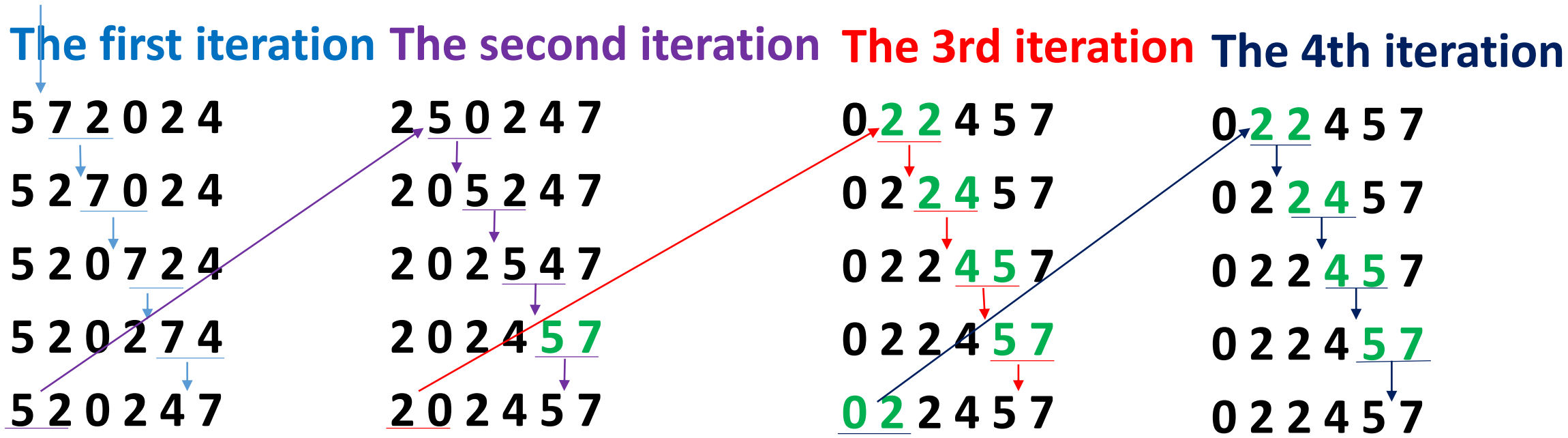
Go to Step 0 and repeat.
(When to stop: if no swapping takes place from Step 0 to Step n-2.)
(Worst case complexity: Repeat n-1 times to finish sorting.)

Bubble sort

Green indicates that no swapping needed for those two elements

- Example: Array with 6 int type elements

5 7 2 0 2 4 (original)



4 swapping in the first iteration

4 swapping in the second iteration

1 swapping in the third iteration

No swapping in the last iteration; stop.

Binary Search

- **Suppose that there is a sorted (sorted according to names) list containing the names of students and their grades. Now given the name of a student, find his/her grade**
  - **You can find the student name in the list in a sequential way (by checking the names in the list one by one), but it is not efficient**
  - **We can use binary search to speed up the search**
- **Binary Search (also called half-interval search)**
  - **Keep reducing the search space by at least half**
  - **Complexity: Assume that there are N elements in the sorted array,**
    - **Worst case: accessing at most floor($\log_2 N$) + 1 elements**
    - **very efficient (suppose that there are around 1 million elements, complexity is around 21)**

Binary Search

- **The algorithm:**

  Suppose that there are n sorted elements in array A, we want to find the index of element with value x in array A. Let the search interval be [min, max]

  Step 0. At the beginning, initialization: min = 0, max = n-1;

  Step 1. Compute the middle point mid = floor ((min+max)/2) ;

  If x == A[mid], done;

  if x < A[mid], max = mid-1;

  if x > A[mid], min = mid+1;

  Step 2. Repeat Step 1.

  (Stop if x == A[mid], or if max < min)

Binary Search

- **In the previous slide, we find only the index of one element in the sorted array**
  - **If there are multiple elements with the same value, once we find the index of one of those elements, we can find the other indices by increasing/decreasing the index we have found.**

Binary Search

- **Example: find the index of 31 in the sorted array A**

index	0	1	2	3	4	5	6	7	8	9
	12	13	17	26	28	31	38	40	44	45

[0, 9] => mid = 4 ; A[4] = 28; A[4] < 31, set min = 4+1 = 5;

	12	13	17	26	28	31	38	40	44	45

[5, 9] => mid = 7; A[7] = 40; A[7] > 31, set max = 7-1 = 6;

	12	13	17	26	28	31	38	40	44	45

[5, 6] => mid = 5; A[5] = 31;