C++ Strings

Creating String Objects

C-string

Array of chars that is null terminated ('\0').

C++ - string

- Object whose string type is defined in the **<string>** file
- has a large repertoire of functions (e.g. length, replace, etc.)

```
char cs[ ] = "Napoleon";  // C-string
string s = "Napoleon";    // C++ - string

cout << s << " has " <<  s.length() << " characters.\n";
s.replace(5, 2,"ia");  //changes s to "Napolian
```


- **Formatted Input:** Stream extraction operator
  - `cin >> stringObject;`
  - the extraction operator >> formats the data that it receives through its input stream; it skips over whitespace
- Unformatted Input: **getline** function for a **string**
  - `getline( cin, s)`
    - does not skip over whitespace
    - delimited by newline
    - reads an entire line of characters into s

```
string s = "ABCDEFG";
getline(cin, s);  //reads entire line of characters into s
char c = s[2];   //assigns 'C' to c
S[4] = '*';      //changes s to "ABCD*FG"
```


- Not necessarily `null` terminated
- `string` is not a pointer, but a class
- Many member functions take start position and length
  - If length argument too large, max chosen

Creating String Objects

```
#include <string>
//string initialization

string s;  //s contains 0 characters
string s1( "Hello" );  //s1 contains 5 characters
string s2 = "Hello"; //s2 contains 5 characters
                     //implicitly calls the constructor

string s3( 8, 'x' );   //s3 contains 8  'x' characters
string s4 = s3;        //s4 contains 8  'x' characters

string s5(s2, 3, 2);  //s5 copies a substring of s2; it contains "lo"
```

string type in the **<string>** header file.

String Objects

C++ strings can be converted to C-strings:

```
string s = "ABCDEFG";
const char* cs = s.c_str();
```

Converts s into the C-string cs.

The **c_str()** function has a return type **const char***

String Objects

The C++ string class also defines a length() function for extracting how many characters are stored in a string.

```
cout << s.length() << endl;
```

Prints 4 for the string s == "Leon"

You can also use the ***subscript operator*** **[ ]** to access individual characters:

e.g. **s[0] = 'N' ;** *//where* **index: 0** to **length-1**

String Objects

C++ strings can be compared using relational operators just like fundamental types:

```
If(s2 < s5)
   cout << "s2 lexicographically precedes s5 \n";

while(s4==s3) //…
```

'B' is lexicographically greater than 'A'

`Sample order: 'A',"Apple", "Banana", "Zest", 'a', "apricot", "leon"`

String Objects

You can also concatenate C++ strings using the + and += operators:

```
string s = "ABCD*FG";
string s2 = "Robot";
string s5 = "Soccer";
string s6  = s + "HIJK";   //changes s6 to "ABCD*FGHIJK

s2 += s5;  //changes s2 to "RobotSoccer"
```

9

String Objects

Substring function: substr()

```
s6 = "ABCD*FGHIJK";
s4 = s6.substr(5, 3); //changes s4 to "FGH"
```

s4 gets a substring of s6, starting at index 5 and taking 3 characters

10

String Objects

erase() and replace() functions:

```
s6 = "ABCD*FGHIJK";
s6.erase(4, 2); //changes s6 to "ABCDGHIJK";

s6.replace(5, 2, "xyz"); //changes s6 to "ABCDGxyzJK";
```

replace 2 characters from s6, starting at index 5, with "xyz"

11

String Objects

find() function
returns the index of the first occurrence of a given substring:

```
string s7 = "Mississippi River basin";   //23 characters
cout << s7.find("si") << endl;   //prints 3
cout << s7.find("so") << endl;  //prints 23, the length of the string
```

If the find() function fails, it returns the length of the string it was searching.

i.e. find() returns 4,294,967,295

12

Assignment

- **Assignment**
  - `s2 = s1;`
    - Makes a separate copy
  - `s2.assign(s1);`
    - Same as `s2 = s1;`
  - `myString.assign(s, start, N);`
    - Copies `N` characters from `s`, beginning at index `start`
  - Individual character assignment
    - `s2[0] = s3[2];`

13

Range-checking

- **Range-checking**
  - `s3.at( index );`
    - Returns character at `index`
    - Can throw an `out_of_range` exception
  - `[]` has no range checking

```
#include <exception>
...

string s = "leon";
try{
   char letter = s.at( 50 );
   cout <<"letter is = " << letter << endl;
}

catch(exception&  e){
          cout << "out_of_range exception:  " <<  e.what() <<  endl;
}
```

14

Concatenation

- **Concatenation**
  - `s3.append( "pet" );`
  - `s3 += "pet";`
    - Both add `"pet"` to end of `s3`
  - `s3.append( s1, start, N );`
    - Appends `N` characters from `s1`, beginning at index `start`

15

Comparing strings

- Overloaded operators
  - `==`, `!=`, `<`, `>`, `<=` and `>=`
  - returns `bool`
- `s1.compare(s2)`
  - returns positive if `s1` is lexicographically greater
    - compares letter by letter
    - `'B'` lexicographically greater than `'A'`
    - `'a'` lexicographically greater than `'A'`
    - `'a'` lexicographically greater than `'Z'`
  - returns negative if less; zero if equal
    - `Sample order: 'A',"Apple", "Banana", "Zest", 'a', "apricot", "leon"`
  - `s1.compare(start, length, s2, start, length)`
    - Compare portions of `s1` and `s2`
  - `s1.compare(start, length, s2)`
    - Compare portion of `s1` with all of `s2`

16

Substrings

- Function **`substr`** gets a substring
  - `s1.substr( start, N );`
  - gets `N` characters, beginning with index `start`
  - returns substring

17

Swapping strings

- `s1.swap(s2);`
  - Switch contents of two strings

18

string Characteristics

- Member functions
  - `s1.size()` and `s1.length()`
    - Number of characters in a string
  - `s1.capacity()`
    - Number of elements that can be stored without reallocation
  - `s1.max_size()`
    - Maximum possible string size
  - `s1.empty()`
    - Returns `true` if empty
  - `s1.resize(newlength)`
    - Resizes string to `newlength`

19

Finding Strings and Characters in a string

- **Find functions**
  - If found, **index** returned
  - If not found, `string::npos` returned
    - Public static constant in class string
  - `s1.find( s2 )`
  - `s1.rfind( s2 )`
    - Searches right-to-left
  - `s1.find_first_of( s2 )`
    - Returns first occurrence of any character in `s2`
    - `Example: s1.find_first_of( "abcd" )`
      - Returns index of first `'a'`, `'b'`, `'c'` or `'d'`

20

Finding Strings and Characters in a string

- **Find functions**
  - `s1.find_last_of( s2 )`
    - Finds last occurrence of any character in `s2`
  - `s1.find_first_not_of( s2 )`
    - Finds first character NOT in `s2`
  - `s1.find_last_not_of( s2 )`
    - Finds last character NOT in `s2`

Replacing Characters in a string

- `s1.erase( start )`
  - Erase from index `start` to end of string, including `start`
- Replace
  - `s1.replace( begin, N, s2)`
    - `begin`: index in `s1` to start replacing
    - `N`: number of characters to replace
    - `s2`: replacement string
  - `s1.replace( begin, N, s2, index, num )`
    - `index`: element in `s2` where replacement comes from
    - `num`: number of elements to use when replacing
  - Replace can overwrite characters

Example

`s1.replace( begin, N, s2, index, num )`

- `begin`: index in `s1` to start replacing
- `N`: number of characters to replace
- `s2`: replacement string
- `index`: element in `s2` where replacement comes from
- `num`: number of elements to use when replacing

```
string str = "this is an example string.";
string str3="sample phrase";

str.replace(19,6, str3, 7, 6);    // "this is an example phrase."
```


Inserting Characters into a string

- `s1.insert( index, s2 )`
  - Inserts `s2` before position `index`
- `s1.insert( index, s2, index2, N );`
  - Inserts substring of `s2` before position `index`
  - Substring is `N` characters, starting at `index2`

Conversion to C-Style char*

- **Conversion functions**
  - `String`s are not necessarily null-terminated
  - `s1.copy( ptr, N, index )`
    - Copies `N` characters **into** the array `ptr`
    - Starts at location `index`
    - Need to null terminate

```
char str[8];

string s2 = "cathode";

s2.copy(str, 5, 2); //copy 5 characters into str
                    //starting at index 2

//strcat(str,"\0"); //does not work
str[5] = '\0';      //this is required

cout << "str = " << str << endl;
cout << "s2 = " << s2 << endl;
```

Output:

str = thode
s2 = cathode

Conversion to C-Style char * Strings

- **Conversion functions**
  - `s1.c_str()`
    - Returns `const char *`
    - Null terminated
    - *e.g.* Useful for filenames:
      ifstream in(`s1.c_str()`);
  - `s1.data()`
    - Returns `const char *`
    - NOT null-terminated

Warning!

- No conversion from `int` or `char`
  - The following definitions could return errors, or warnings only, but then would cause the program to crash afterwards
    - `string error1 = 'c';`
    - `string error2( 'u' );`
    - `string error3 = 22;`
    - `string error4( 8 );`
  - However, it can be assigned one `char` after its declaration:
    - `s = 'n';`

String Stream Processing

- allows a string to be used as an internal file
- useful for buffering input and output

- I/O of strings to and from memory
  - Called in-memory I/O or string stream processing
  - Classes
    - `istringstream` // input from string
    - `ostringstream` // output to a string
    - `stringstream( string )` // most useful
    - `Requires` `<sstream>` and `<iostream>` headers
  - Use string formatting to save data to memory

Output String Stream

```
ostringstream  oss;

int n = 44;
float x = 3.14;

oss << "Hello!\t" << n << '\t' << x;
string  s = oss.str();

cout << endl << s << endl;
```

Serves as a conduit to an anonymous string which can be read with the built-in oss.str() function that is bound to the oss object

Remember sprintf()?, how does it compare to this one?

29

Input String Stream

```
const string buffer = oss.str();
istringstream  iss(buffer);

string word;
int m;
float y;

iss >> word >> m >> y;

s = iss.str();
cout << endl << s << endl;

cout << "word = " << word << endl;
cout << "m = " << m << endl;
cout << "y = " << y << endl;
```

iss is defined and bound to buffer

Contents of **buffer** can be accessed as elements of a string, or by formatted input through the **iss** object.

Remember sscanf()?, how does it compare to this one?

All extractions from iss will come from the contents of buffer, as if it were an external file.

30

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <sstream>
using namespace std;
int main(){

  string s1("mydata.txt");
  ifstream in( s1.c_str() );

  char buffer[1024];
  while( in.getline( buffer, 1024 ) ){

    string stemp( buffer );
    cout << "Line is:" << stemp << endl;

    if( stemp[0] != '#' ){
      stringstream stris( stemp );
      double d1, d2;
      stris >> d1 >> d2;
      cout << d1 << "," << d2 << endl;
    }
    cout << endl;
  }
  in.close();
  return 0;
}
```

Using string example

- **Input file:**

```
1.0   2.0
1.1    2.4
1.8  2.8
#1.34  2.99
1.4     8.99
```

- **Example Output:**

```
Line is:1.0   2.0
1,2

Line is:1.1    2.4
1.1,2.4

Line is:1.8  2.8
1.8,2.8

Line is:#1.34  2.99

Line is:1.4     8.99
1.4,8.99
```

(or could use strtok, C String tokenizers)

31

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <sstream>
using namespace std;
int main(){

  string s1("mydata.txt");
  ifstream in( s1.c_str() );

  char buffer[1024];
  while( in.getline( buffer, 1024 ) ){

    string stemp( buffer );
    cout << "Line is:" << stemp << endl;

    if( stemp[0] != '#' ){
      stringstream stris( stemp );
      double d1, d2;
      stris >> d1 >> d2;
      cout << d1 << "," << d2 << endl;
    }
    cout << endl;
  }
  in.close();
  return 0;
}
```

Alternatively: (no C-style char*)

```
int main(){

  string s1("mydata.txt");
  ifstream in( s1.c_str() );

  string buffer;
  while(getline( in, buffer ) ){

    cout << "Line is:" << buffer << endl;

    if( buffer[0]  !=  '#' ){
      istringstream stris( buffer );
      double d1, d2;
      stris >> d1 >> d2;
      cout << "data: " << d1 << "," << d2 << endl;
    }
    cout << endl;
  }
  in.close();
  return 0;
}
```

32

Summary

C++ strings are safer and easier to use than C string.

33

Method	Use

append(char *pt);
append(char *pt, size_t count);
append(string &str, size_t offset,size_t count);
append(string &str);
append(size_t count, char ch);
append(InputIterator Start, InputIterator End);
	Appends characters to a string from C-style strings, char's or other string objects.

at(size_t offset);	Returns a reference to the character at the specified position. Differs from the subscript operator, [], in that bounds are checked.
begin();	Returns an iterator to the start of the string.
*c_str();	Returns a pointer to C-style string version of the contents of the string.
clear();	Erases the entire string.
copy(char *cstring, size_t count, size_t offset);	Copies "count" characters into a C-style string starting at offset.
empty();	Test whether a string is empty.
end();	Returns an iterator to one past the end of the string.

erase(iterator first, iterator last);
erase(iterator it);
erase(size_t pos, size_t count);
	Erases characters from the specified positions.

34

Method	Use

find(char ch,size_t offset = 0);
find(char *pt,size_t offset = 0);
find(string &str,size_t offset = 0);
	Returns the index of the first character of the substring when found. Otherwise, the special value "npos" is returned.

find_first_not_of();	Same sets of arguments as find. Finds the index of the first character that is not in the search string.
find_first_of();	Same sets of arguments as find. Finds the index of the first character that is in the search string.
find_last_not_of();	Same sets of arguments as find. Finds the index of the last character that is not in the search string.
find_last_of();	Same sets of arguments as find. Finds the index of the last character that is in the search string.

insert(size_t pos, char *ptr);
insert(size_t pos, string &str);
insert(size_t pos, size_t count, char ch);
insert(iterator it, InputIterator start, InputIterator end);
	Inserts characters at the specified position.

push_back(char ch);	Inserts a character at the end of the string.

replace(size_t pos, size_t count, char *pt);
replace(size_t pos, size_t count, string &str);
replace(iterator first, iterator last, char *pt);
replace(iterator first, iterator last, string &str);
	Replaces elements in a string with the specified characters. The range can be specified by a start position and a number of elements to replace, or by using iterators.

size();	Returns the number of elements in a string.

35