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”**

STRINGS

- Not necessarily **null** terminated
- **string** is not a pointer, but a class

Creating String Objects

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

```
#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 ”l0”
```

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 == “Blue”

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”

String Objects

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

```
string s = “ABCD*FG”;
string s2 = “MIT”;
string s5 = “Manipal”;
string s6  = s + “HIJK”;   //changes s6 to “ABCD*FGHIJK

s2 += s5;  //changes s2 to “MITManipal”
```

String Objects

Substring function: **substr()**

s6 = “ABCD***FGH**IJK”;

s4 = s6.**substr(**5, 3); //changes

s4 to “FGH”

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

String Objects

erase() and replace() functions:

s6 = “ABCD***F**GHIJK”;

s6**.erase(**4, 2**);** //changes s6 to “ABCDG**HI**JK”;

s6.**replace(**5, 2, “xyz”**);** //changes s6 to “ABCDG**xyz**JK”;

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

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.

Assignment in strings

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];

Range-checking

s3.at(index); Returns character at **index**

Can throw an **out_of_range** exception

[ ] has no range checking

```
 #include <exception>
...

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

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

Concatenation

s3.append(“MIT");
s3 += “MIT";
s3.append(s1, start, N);

Both add **“MIT"** to end of **s3**
Appends **N** characters from **s1**, beginning at index **start**

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", "pear"**
  - **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**

Substrings

Function **substr** gets a substring

s1.substr(start, N);

gets **N** characters, beginning with index **start and** returns substring

Swapping strings

s1.swap(s2);

switches contents of two strings

Finding Strings and Characters in a string

Find functions: If found, **index** returned
If not found, **string::npos** returned

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'**

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

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

Strings 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

Example: 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';
```