Chapter 18

STRINGS IN DEPTH

***Listing 18-1. Illustrating String Storage***

**//: C18:StringStorage.cpp**

**#include <string>**

**#include <iostream>**

**using namespace std;**

**int main() {**

**string s1("12345");**

**// Set the iterator indicate the first element**

**string::iterator it = s1.begin();**

**// This may copy the first to the second or**

**// use reference counting to simulate a copy**

**string s2 = s1;**

**// Either way, this statement may ONLY modify first**

**\*it = '0';**

**cout << "s1 = " << s1 << endl;**

**cout << "s2 = " << s2 << endl;**

**} ///:~**

***Listing 18-2. Illustrating String Features***

**//: C18:SmallString.cpp**

**#include <string>**

**using namespace std;**

**int main() {**

**string imBlank;**

**string heyMom("Where are my socks?");**

**string standardReply = "Beamed into deep "**

**"space on wide angle dispersion?";**

**string useThisOneAgain(standardReply);**

**} ///:~**

***Listing 18-3. Illustrating More String Features***

**//: C18:SmallString2.cpp**

**#include<string>**

**#include<iostream>**

**using namespace std;**

**int main() {**

**string s1("What is the sound of one clam napping?");**

**string s2("Anything worth doing is worth overdoing.");**

**string s3("I saw Elvis in a UFO");**

**// Copy the first 8 chars:**

**string s4(s1, 0, 8);**

**cout << s4 << endl;**

**// Copy 6 chars from the middle of the source:**

**string s5(s2, 15, 6);**

**cout << s5 << endl;**

**// Copy from middle to end:**

**string s6(s3, 6, 15);**

**cout << s6 << endl;**

**// Copy many different things:**

**string quoteMe = s4 + "that" +**

**// substr() copies 10 chars at element 20**

**s1.substr(20, 10) + s5 +**

**// substr() copies up to either 100 char**

**// or eos starting at element 5**

**"with" + s3.substr(5, 100) +**

**// OK to copy a single char this way**

**s1.substr(37, 1);**

**cout << quoteMe << endl;**

**} ///:~**

***Listing 18-4. Illustrating String Iterators***

**//: C18:StringIterators.cpp**

**#include <string>**

**#include <iostream>**

**#include <cassert>**

**using namespace std;**

**int main() {**

**string source("xxx");**

**string s(source.begin(), source.end());**

**assert(s == source);**

**} ///:~**

***Listing 18-5. Illustrating Initialization of Strings***

**//: C18:UhOh.cpp**

**#include <string>**

**#include <cassert>**

**using namespace std;**

**int main() {**

**// Error: no single char inits**

**//! string nothingDoing1('a');**

**// Error: no integer inits**

**//! string nothingDoing2(0x37);**

**// The following is legal:**

**string okay(5, 'a');**

**assert(okay == string("aaaaa"));**

**} ///:~**

***Listing 18-6. Illustrating Reallocation of Storage as per String Size***

**//: C18:StrSize.cpp**

**#include <string>**

**#include <iostream>**

**using namespace std;**

**int main() {**

**string bigNews("I saw Elvis in a UFO. ");**

**cout << bigNews << endl;**

**// How much data have we actually got?**

**cout << "Size = " << bigNews.size() << endl;**

**// How much can we store without reallocating?**

**cout << "Capacity = " << bigNews.capacity() << endl;**

**// Insert this string in bigNews immediately**

**// before bigNews[1]:**

**bigNews.insert(1, " thought I");**

**cout << bigNews << endl;**

**cout << "Size = " << bigNews.size() << endl;**

**cout << "Capacity = " << bigNews.capacity() << endl;**

**// Make sure that there will be this much space**

**bigNews.reserve(500);**

**// Add this to the end of the string:**

**bigNews.append("I've been working too hard.");**

**cout << bigNews << endl;**

**cout<< "Size = " << bigNews.size() << endl;**

**cout << "Capacity = " << bigNews.capacity() << endl;**

**} ///:~**

***Listing 18-7. Illustrating Replacement of String Characters***

**//: C18:StringReplace.cpp**

**// Simple find-and-replace in strings.**

**#include <cassert>**

**#include <string>**

**using namespace std;**

**int main() {**

**string s("A piece of text");**

**string tag("$tag$");**

**s.insert(8, tag + ' ');**

**assert(s == "A piece $tag$ of text");**

**int start = s.find(tag);**

**assert(start == 8);**

**assert(tag.size() == 5);**

**s.replace(start, tag.size(), "hello there");**

**assert(s == "A piece hello there of text");**

**} ///:~**

***Listing 18-8. Illustrating A More Complete Demonstration of replace( )***

**//: C18:Replace.cpp**

**#include <cassert>**

**#include <cstddef>  // For size\_t**

**#include <string>**

**using namespace std;**

**void replaceChars(string& modifyMe,**

**const string& findMe, const string& newChars) {**

**// Look in modifyMe for the "find string"**

**// starting at position 0:**

**size\_t i = modifyMe.find(findMe, 0);**

**// Did we find the string to replace?**

**if(i != string::npos)**

**// Replace the find string with newChars:**

**modifyMe.replace(i, findMe.size(), newChars);**

**}**

**int main() {**

**string bigNews = "I thought I saw Elvis in a UFO. "**

**"I have been working too hard.";**

**string replacement("wig");**

**string findMe("UFO");**

**// Find "UFO" in bigNews and overwrite it:**

**replaceChars(bigNews, findMe, replacement);**

**assert(bigNews == "I thought I saw Elvis in a "**

**"wig. I have been working too hard.");**

**} ///:~**

***Listing 18-9. Illustrating String Replacement and Growth***

**//: C18:ReplaceAndGrow.cpp**

**#include<cassert>**

**#include<string>**

**using namespace std;**

**int main() {**

**string bigNews("I have been working the grave.");**

**string replacement("yard shift.");**

**// The first argument says "replace chars**

**// beyond the end of the existing string":**

**bigNews.replace(bigNews.size() - 1,**

**replacement.size(), replacement);**

**assert(bigNews == "I have been working the "**

**"graveyard shift.");**

**} ///:~**

***Listing 18-10. Illustrating ReplaceAll***

**//: C18:ReplaceAll.h**

**#ifndef REPLACEALL\_H**

**#define REPLACEALL\_H**

**#include <string>**

**std::string& replaceAll(std::string& context,**

**const std::string& from, const std::string& to);**

**#endif // REPLACEALL\_H ///:~**

**//: C18:ReplaceAll.cpp {O}**

**#include <cstddef>**

**#include "ReplaceAll.h"// To be INCLUDED from Header FILE above**

**using namespace std;**

**string& replaceAll(string& context, const string& from,**

**const string& to) {**

**size\_t lookHere = 0;**

**size\_t foundHere;**

**while((foundHere = context.find(from, lookHere))**

**!= string::npos) {**

**context.replace(foundHere, from.size(), to);**

**lookHere = foundHere + to.size();**

**}**

**return context;**

**} ///:~**

***Listing 18-11. Illustrating Test of The ReplaceAll in Listing 18-10***

**//: C18:ReplaceAllTest.cpp**

**//{L} ../C18/ReplaceAll**

**#include <cassert>**

**#include <iostream>**

**#include <string>**

**#include "ReplaceAll.h"**

**using namespace std;**

**int main() {**

**string text = "a man, a plan, a canal, Panama";**

**replaceAll(text, "an", "XXX");**

**assert(text == "a mXXX, a plXXX, a cXXXal, PXXXama");**

**} ///:~**

***Listing 18-12. Illustrating String Character Replacement***

**//: C18:StringCharReplace.cpp**

**#include <algorithm>**

**#include <cassert>**

**#include <string>**

**using namespace std;**

**int main() {**

**string s("aaaXaaaXXaaXXXaXXXXaaa");**

**replace(s.begin(), s.end(), 'X', 'Y');**

**assert(s == "aaaYaaaYYaaYYYaYYYYaaa");**

**} ///:~**

***Listing 18-13. Illustrating Addition of Strings***

**//: C18:AddStrings.cpp**

**#include <string>**

**#include <cassert>**

**using namespace std;**

**int main() {**

**string s1("This ");**

**string s2("That ");**

**string s3("The other ");**

**// operator+ concatenates strings**

**s1 = s1 + s2;**

**assert(s1 == "This That ");**

**// Another way to concatenates strings**

**s1 += s3;**

**assert(s1 == "This That The other ");**

**// You can index the string on the right**

**s1 += s3 + s3[4] + "ooh lama";**

**assert(s1 == "This That The other The other oooh lala");**

**} ///:~**

***Listing 18-14. Illustrating The Sieve of Eratosthenes***

***(to find prime numbers < 50)***

**//: C18:Sieve.cpp**

**#include <string>**

**#include <iostream>**

**using namespace std;**

**int main() {**

**// Create a 50 char string and set each**

**// element to 'P' for Prime**

**string sieveChars(50, 'P');**

**// By definition neither 0 nor 1 is prime.**

**// Change these elements to "N" for Not Prime**

**sieveChars.replace(0, 2, "NN");**

**// Walk through the array:**

**for(int i = 2;**

**i <= (sieveChars.size() / 2) - 1; i++)**

**// Find all the factors:**

**for(int factor = 2;**

**factor \* i < sieveChars.size();factor++)**

**sieveChars[factor \* i] = 'N';**

**cout << "Prime:" << endl;**

**// Return the index of the first 'P' element:**

**int j = sieveChars.find('P');**

**// While not at the end of the string:**

**while(j != sieveChars.npos) {**

**// If the element is P, the index is a prime**

**cout << j << " ";**

**// Move past the last prime**

**j++;**

**// Find the next prime**

**j = sieveChars.find('P', j);**

**}**

**cout << "\n Not prime:" << endl;**

**// Find the first element value not equal P:**

**j = sieveChars.find\_first\_not\_of('P');**

**while(j != sieveChars.npos) {**

**cout << j << " ";**

**j++;**

**j = sieveChars.find\_first\_not\_of('P', j);**

**}**

**} ///:~**

***Listing 18-15. Using find( ) to Detect a Sequence of Characters***

**//: C18:Find.cpp**

**// Find a group of characters in a string**

**#include <string>**

**#include <iostream>**

**using namespace std;**

**int main() {**

**string chooseOne("Eenie, meenie, miney, mo");**

**int i = chooseOne.find("een");**

**while(i != string::npos) {**

**cout << i << endl;**

**i++;**

**i = chooseOne.find("een", i);**

**}**

**} ///:~**

***Listing 18-16. Using find( ) for A Case-Insensitive Search***

**//: C18:NewFind.cpp**

**#include <string>**

**#include <iostream>**

**using namespace std;**

**// Make an uppercase copy of s:**

**string upperCase(string& s) {**

**char\* buf = new char[s.length()];**

**s.copy(buf, s.length());**

**for(int i = 0; i < s.length(); i++)**

**buf[i] = toupper(buf[i]);**

**string r(buf, s.length());**

**delete buf;**

**return r;**

**}**

**// Make a lowercase copy of s:**

**string lowerCase(string& s) {**

**char\* buf = new char[s.length()];**

**s.copy(buf, s.length());**

**for(int i = 0; i < s.length(); i++)**

**buf[i] = tolower(buf[i]);**

**string r(buf, s.length());**

**delete buf;**

**return r;**

**}**

**int main() {**

**string chooseOne("Eenie, meenie, miney, mo");**

**cout << chooseOne << endl;**

**cout << upperCase(chooseOne) << endl;**

**cout << lowerCase(chooseOne) << endl;**

**// Case sensitive search**

**int i = chooseOne.find("een");**

**while(i != string::npos) {**

**cout << i << endl;**

**i++;**

**i = chooseOne.find("een", i);**

**}**

**// Search lowercase:**

**string lcase = lowerCase(chooseOne);**

**cout << lcase << endl;**

**i = lcase.find("een");**

**while(i != lcase.npos) {**

**cout << i << endl;**

**i++;**

**i = lcase.find("een", i);**

**}**

**// Search uppercase:**

**string ucase = upperCase(chooseOne);**

**cout << ucase << endl;**

**i = ucase.find("EEN");**

**while(i != ucase.npos) {**

**cout << i << endl;**

**i++;**

**i = ucase.find("EEN", i);**

**}**

**} ///:~**

***Listing 18-17. Finding in Reverse Using rfind( )***

**//: C18:Rparse.cpp**

**// Reverse the order of words in a string**

**#include <string>**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**int main() {**

**// The ';' characters will be delimiters**

**string s("now.;sense;make;to;going;is;This");**

**cout << s << endl;**

**// To store the words:**

**vector<string> strings;**

**// The last element of the string:**

**int last = s.size();**

**// The beginning of the current word:**

**int current = s.rfind(';');**

**// Walk backward through the string:**

**while(current != string::npos){**

**// Push each word into the vector.**

**// Current is incremented before copying to**

**// avoid copying the delimiter.**

**strings.push\_back(**

**s.substr(++current,last - current));**

**// Back over the delimiter we just found,**

**// and set last to the end of the next word**

**current -= 2;**

**last = current;**

**// Find the next delimiter**

**current = s.rfind(';', current);**

**}**

**// Pick up the first word - it's not**

**// preceded by a delimiter**

**strings.push\_back(s.substr(0, last - current));**

**// Print them in the new order:**

**for(int j = 0; j < strings.size(); j++)**

**cout << strings[j] << " ";**

**} ///:~**

***Listing 18-18. Stripping Whitespaces, that is, Trimming a String***

**//: C18:trim.h**

**#ifndef TRIM\_H**

**#define TRIM\_H**

**#include <string>**

**// General tool to strip spaces from both ends:**

**inline std::string trim(const std::string& s) {**

**if(s.length() == 0)**

**return s;**

**int b = s.find\_first\_not\_of(" \t");**

**int e = s.find\_last\_not\_of(" \t");**

**if(b == -1) // No non-spaces**

**return "";**

**return std::string(s, b, e - b + 1);**

**}**

**#endif // TRIM\_H ///:~**

***Listing 18-19. Testing Out “trim.h” in Listing 18-18***

**//: C18:TrimTest.cpp**

**#include "trim.h" // To be INCLUDED from Header FILE above**

**#include <iostream>**

**using namespace std;**

**string s[] = {**

**" \t abcdefghijklmnop \t ",**

**"abcdefghijklmnop \t ",**

**" \t abcdefghijklmnop",**

**"a", "ab", "abc", "a b c",**

**" \t a b c \t ", " \t a \t b \t c \t ",**

**"", // Must also test the empty string**

**};**

**void test(string s) {**

**cout << "[" << trim(s) << "]" << endl;**

**}**

**int main() {**

**for(int i = 0; i < sizeof s / sizeof \*s; i++)**

**test(s[i]);**

**} ///:~**

***Listing 18-20. Illustrating An HTML Stripper Using erase( )***

**//: C18:HTMLStripper.cpp {RunByHand}**

**//{L} ../C18/ReplaceAll**

**// Filter to remove html tags and markers.**

**#include <cassert>**

**#include <cmath>**

**#include <cstddef>**

**#include <fstream>**

**#include <iostream>**

**#include <string>**

**#include "ReplaceAll.h" // SEE Above**

**#include "../require.h" // To be INCLUDED from *Chapter 9***

**using namespace std;**

**string& stripHTMLTags(string& s) {**

**static bool inTag = false;**

**bool done = false;**

**while(!done) {**

**if(inTag) {**

**// The previous line started an HTML tag**

**// but didn't finish. Must search for '>'.**

**size\_t rightPos = s.find('>');**

**if(rightPos != string::npos) {**

**inTag = false;**

**s.erase(0, rightPos + 1);**

**}**

**else {**

**done = true;**

**s.erase();**

**}**

**}**

**else {**

**// Look for start of tag:**

**size\_t leftPos = s.find('<');**

**if(leftPos != string::npos) {**

**// See if tag close is in this line:**

**size\_t rightPos = s.find('>');**

**if(rightPos == string::npos) {**

**inTag = done = true;**

**s.erase(leftPos);**

**}**

**else**

**s.erase(leftPos, rightPos - leftPos + 1);**

**}**

**else**

**done = true;**

**}**

**}**

**// Remove all special HTML characters**

**replaceAll(s, "&lt;", "<");**

**replaceAll(s, "&gt;", ">");**

**replaceAll(s, "&amp;", "&");**

**replaceAll(s, "&nbsp;", " ");**

**// Etc...**

**return s;**

**}**

**int main(int argc, char\* argv[]) {**

**requireArgs(argc, 1,**

**"usage: HTMLStripper InputFile");**

**ifstream in(argv[1]);**

**assure(in, argv[1]);**

**string s;**

**while(getline(in, s))**

**if(!stripHTMLTags(s).empty())**

**cout << s << endl;**

**} ///:~**

***Listing 18-21. Illustrating Comparison of Strings***

**//: C18:CompStr.cpp**

**#include <string>**

**#include <iostream>**

**using namespace std;**

**int main() {**

**// Strings to compare**

**string s1("This ");**

**string s2("That ");**

**for(int i = 0; i < s1.size() &&**

**i < s2.size(); i++)**

**// See if the string elements are the same:**

**if(s1[i] == s2[i])**

**cout << s1[i] << " " << i << endl;**

**// Use the string inequality operators**

**if(s1 != s2) {**

**cout << "Strings aren't the same:" << " ";**

**if(s1 > s2)**

**cout << "s1 is > s2" << endl;**

**else**

**cout << "s2 is > s1" << endl;**

**}**

**} ///:~**

***Listing 18-22. Illustrating Equivalence in String Comparison***

**//: C18:Equivalence.cpp**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**string s2("That"), s1("This");**

**// The lvalue is a quoted literal**

**// and the rvalue is a string:**

**if("That" == s2)**

**cout << "A match" << endl;**

**// The left operand is a string and the right is**

**// a pointer to a C-style null terminated string:**

**if(s1 != s2.c\_str())**

**cout << "No match" << endl;**

**} ///:~**

***Listing 18-23. Comparing Complete Strings***

**//: C18:Compare.cpp**

**// Demonstrates compare() and swap().**

**#include <cassert>**

**#include <string>**

**using namespace std;**

**int main() {**

**string first("This");**

**string second("That");**

**assert(first.compare(first) == 0);**

**assert(second.compare(second) == 0);**

**// Which is lexically greater?**

**assert(first.compare(second) > 0);**

**assert(second.compare(first) < 0);**

**first.swap(second);**

**assert(first.compare(second) < 0);**

**assert(second.compare(first) > 0);**

**} ///:~**

***Listing 18-24. Comparing a Subset of Characters in One or Both Strings***

**//: C18:Compare2.cpp**

**// Illustrate overloaded compare().**

**#include <cassert>**

**#include <string>**

**using namespace std;**

**int main() {**

**string first("This is a day that will live in infamy");**

**string second("I don't believe that this is what "**

**"I signed up for");**

**// Compare "his is" in both strings:**

**assert(first.compare(1, 7, second, 22, 7) == 0);**

**// Compare "his is a" to "his is w":**

**assert(first.compare(1, 9, second, 22, 9) < 0);**

**} ///:~**

***Listing 18-25. Demonstrating Similarity between String Indexing with [ ] and at( )***

**//: C18:StringIndexing.cpp**

**#include <cassert>**

**#include <string>**

**using namespace std;**

**int main() {**

**string s("1234");**

**assert(s[1] == '2');**

**assert(s.at(1) == '2');**

**} ///:~**

***Listing 18-26. Demonstrating Difference between String Indexing with [ ] and at( )***

**//: C18:BadStringIndexing.cpp**

**#include <exception>**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**string s("1234");**

**// at() saves you by throwing an exception:**

**try {**

**s.at(5);**

**} catch(exception& e) {**

**cerr << e.what() << endl;**

**}**

**} ///:~**

***Listing 18-27. Developing ichar\_traits***

**//: C18:ichar\_traits.h**

**// Creating your own character traits.**

**#ifndef ICHAR\_TRAITS\_H**

**#define ICHAR\_TRAITS\_H**

**#include <cassert>**

**#include <cctype>**

**#include <cmath>**

**#include <cstddef>**

**#include <ostream>**

**#include <string>**

**using std::allocator;**

**using std::basic\_string;**

**using std::char\_traits;**

**using std::ostream;**

**using std::size\_t;**

**using std::string;**

**using std::toupper;**

**using std::tolower;**

**struct ichar\_traits : char\_traits<char> {**

**// We'll only change character-by-**

**// character comparison functions**

**static bool eq(char c1st, char c2nd) {**

**return toupper(c1st) == toupper(c2nd);**

**}**

**static bool ne(char c1st, char c2nd) {**

**return !eq(c1st, c2nd);**

**}**

**static bool lt(char c1st, char c2nd) {**

**return toupper(c1st) < toupper(c2nd);**

**}**

**static int**

**compare(const char\* str1, const char\* str2, size\_t n) {**

**for(size\_t i = 0; i < n; ++i) {**

**if(str1 == 0)**

**return -1;**

**else if(str2 == 0)**

**return 1;**

**else if(tolower(\*str1) < tolower(\*str2))**

**return -1;**

**else if(tolower(\*str1) > tolower(\*str2))**

**return 1;**

**assert(tolower(\*str1) == tolower(\*str2));**

**++str1; ++str2; // Compare the other chars**

**}**

**return 0;**

**}**

**static const char\***

**find(const char\* s1, size\_t n, char c) {**

**while(n-- > 0)**

**if(toupper(\*s1) == toupper(c))**

**return s1;**

**else**

**++s1;**

**return 0;**

**}**

**};**

**typedef basic\_string<char, ichar\_traits> istring;**

**inline ostream& operator<<(ostream& os, const istring& s) {**

**return os << string(s.c\_str(), s.length());**

**}**

**#endif // ICHAR\_TRAITS\_H ///:~*Listing 18-28. Implementing the header file in Listing 18-27***

**//: C18:ICompare.cpp**

**#include <cassert>**

**#include <iostream>**

**#include "ichar\_traits.h"// To be INCLUDED from Header FILE // above**

**using namespace std;**

**int main() {**

**// The same letters except for case:**

**istring first = "tHis";**

**istring second = "ThIS";**

**cout << first << endl;**

**cout << second << endl;**

**assert(first.compare(second) == 0);**

**assert(first.find('h') == 1);**

**assert(first.find('I') == 2);**

**assert(first.find('x') == string::npos);**

**} ///:~**

***Listing 18-29. Developing Wide-Character Version of ichar\_traits***

**//: C18:iwchar\_traits.h {-g++}**

**// Creating your own wide-character traits.**

**#ifndef IWCHAR\_TRAITS\_H**

**#define IWCHAR\_TRAITS\_H**

**#include <cassert>**

**#include <cmath>**

**#include <cstddef>**

**#include <cwctype>**

**#include <ostream>**

**#include <string>**

**using std::allocator;**

**using std::basic\_string;**

**using std::char\_traits;**

**using std::size\_t;**

**using std::towlower;**

**using std::towupper;**

**using std::wostream;**

**using std::wstring;**

**struct iwchar\_traits : char\_traits<wchar\_t> {**

**// We'll only change character-by-**

**// character comparison functions**

**static bool eq(wchar\_t c1st, wchar\_t c2nd) {**

**return towupper(c1st) == towupper(c2nd);**

**}**

**static bool ne(wchar\_t c1st, wchar\_t c2nd) {**

**return towupper(c1st) != towupper(c2nd);**

**}**

**static bool lt(wchar\_t c1st, wchar\_t c2nd) {**

**return towupper(c1st) < towupper(c2nd);**

**}**

**static int compare(**

**const wchar\_t\* str1, const wchar\_t\* str2, size\_t n) {**

**for(size\_t i = 0; i < n; i++) {**

**if(str1 == 0)**

**return -1;**

**else if(str2 == 0)**

**return 1;**

**else if(towlower(\*str1) < towlower(\*str2))**

**return -1;**

**else if(towlower(\*str1) > towlower(\*str2))**

**return 1;**

**assert(towlower(\*str1) == towlower(\*str2));**

**++str1; ++str2; // Compare the other wchar\_ts**

**}**

**return 0;**

**}**

**static const wchar\_t\***

**find(const wchar\_t\* s1, size\_t n, wchar\_t c) {**

**while(n-- > 0)**

**if(towupper(\*s1) == towupper(c))**

**return s1;**

**else**

**++s1;**

**return 0;**

**}**

**};**

**typedef basic\_string<wchar\_t, iwchar\_traits> iwstring;**

**inline wostream& operator<<(wostream& os,**

**const iwstring& s) {**

**return os << wstring(s.c\_str(), s.length());**

**}**

**#endif // IWCHAR\_TRAITS\_H  ///:~*Listing 18-30. Testing Out the header file Developed in Listing 18-29***

**//: C18:IWCompare.cpp {-g++}**

**#include <cassert>**

**#include <iostream>**

**#include "iwchar\_traits.h" // To be INCLUDED from Header FILE**

**// above**

**using namespace std;**

**int main() {**

**// The same letters except for case:**

**iwstring wfirst = L"tHis";**

**iwstring wsecond = L"ThIS";**

**wcout << wfirst << endl;**

**wcout << wsecond << endl;**

**assert(wfirst.compare(wsecond) == 0);**

**assert(wfirst.find('h') == 1);**

**assert(wfirst.find('I') == 2);**

**assert(wfirst.find('x') == wstring::npos);**

**} ///:~**

 Unfortunately, some compilers still do not provide robust support for wide characters.

# [A String Application](file:///C:\\Documents%20and%20Settings\\cair9582848603\\Local%20Settings\\Temp\\Rar$EXa0.698\\html\\TicV2.html" \l "_TocRef15743863)

If you’ve looked at the sample code in this book closely, you’ve noticed that certain tokens in the comments surround the code. These are used by a Python program that was written to extract the code into files and set up makefiles for building the code. For example, a double slash followed by a colon at the beginning of a line denotes the first line of a source file. The rest of the line contains information describing the file’s name and location and whether it should be only compiled rather than

***Listing 18-31. Illustrating Extraction of All the Source Code in the Book***

**//: C18:ExtractCode.cpp {-edg} {RunByHand}**

**// Extracts code from text.**

**#include <cassert>**

**#include <cstddef>**

**#include <cstdio>**

**#include <cstdlib>**

**#include <fstream>**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**// Legacy non-standard C header for mkdir()**

**#if defined(\_\_GNUC\_\_) || defined(\_\_MWERKS\_\_)**

**#include <sys/stat.h>**

**#elif defined(\_\_BORLANDC\_\_) || defined(\_MSC\_VER) \**

**|| defined(\_\_DMC\_\_)**

**#include <direct.h>**

**#else**

**#error Compiler not supported**

**#endif**

**// Check to see if directory exists**

**// by attempting to open a new file**

**// for output within it.**

**bool exists(string fname) {**

**size\_t len = fname.length();**

**if(fname[len-1] != '/' && fname[len-1] != '\\')**

**fname.append("/");**

**fname.append("000.tmp");**

**ofstream outf(fname.c\_str());**

**bool existFlag = outf;**

**if(outf) {**

**outf.close();**

**remove(fname.c\_str());**

**}**

**return existFlag;**

**}**

**int main(int argc, char\* argv[]) {**

**// See if input file name provided**

**if(argc == 1) {**

**cerr << "usage: extractCode file [dir]" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**// See if input file exists**

**ifstream inf(argv[1]);**

**if(!inf) {**

**cerr << "error opening file: " << argv[1] << endl;**

**exit(EXIT\_FAILURE);**

**}**

**// Check for optional output directory**

**string root("./"); // current is default**

**if(argc == 3) {**

**// See if output directory exists**

**root = argv[2];**

**if(!exists(root)) {**

**cerr << "no such directory: " << root << endl;**

**exit(EXIT\_FAILURE);**

**}**

**size\_t rootLen = root.length();**

**if(root[rootLen-1] != '/' && root[rootLen-1] != '\\')**

**root.append("/");**

**}**

**// Read input file line by line**

**// checking for code delimiters**

**string line;**

**bool inCode = false;**

**bool printDelims = true;**

**ofstream outf;**

**while(getline(inf, line)) {**

**size\_t findDelim = line.find("//" "/:~");**

**if(findDelim != string::npos) {**

**// Output last line and close file**

**if(!inCode) {**

**cerr << "Lines out of order" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**assert(outf);**

**if(printDelims)**

**outf << line << endl;**

**outf.close();**

**inCode = false;**

**printDelims = true;**

**} else {**

**findDelim = line.find("//" ":");**

**if(findDelim == 0) {**

**// Check for '!' directive**

**if(line[3] == '!') {**

**printDelims = false;**

**++findDelim; // To skip '!' for next search**

**}**

**// Extract subdirectory name, if any**

**size\_t startOfSubdir =**

**line.find\_first\_not\_of(" \t", findDelim+3);**

**findDelim = line.find(':', startOfSubdir);**

**if(findDelim == string::npos) {**

**cerr << "missing filename information\n" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**string subdir;**

**if(findDelim > startOfSubdir)**

**subdir = line.substr(startOfSubdir,**

**findDelim - startOfSubdir);**

**// Extract file name (better be one!)**

**size\_t startOfFile = findDelim + 1;**

**size\_t endOfFile =**

**line.find\_first\_of(" \t", startOfFile);**

**if(endOfFile == startOfFile) {**

**cerr << "missing filename" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**// We have all the pieces; build fullPath name**

**string fullPath(root);**

**if(subdir.length() > 0)**

**fullPath.append(subdir).append("/");**

**assert(fullPath[fullPath.length()-1] == '/');**

**if(!exists(fullPath))**

**#if defined(\_\_GNUC\_\_) || defined(\_\_MWERKS\_\_)**

**mkdir(fullPath.c\_str(), 0); // Create subdir**

**#else**

**mkdir(fullPath.c\_str()); // Create subdir**

**#endif**

**fullPath.append(line.substr(startOfFile,**

**endOfFile - startOfFile));**

**outf.open(fullPath.c\_str());**

**if(!outf) {**

**cerr << "error opening " << fullPath**

**<< " for output" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**inCode = true;**

**cout << "Processing " << fullPath << endl;**

**if(printDelims)**

**outf << line << endl;**

**}**

**else if(inCode) {**

**assert(outf);**

**outf << line << endl; // Output middle code line**

**}**

**}**

**}**

**exit(EXIT\_SUCCESS);**

**} ///:~**