How to: Use the Class `string` C

Experience shows that it is impossible to design the perfect **string**. People's taste, expectations, and needs differ too much for that. So, the standard library `string` isn't ideal. I would have made some design decisions differently, and so would you.
Bjarne Stroustrup
The C++ Programming Language, Third Edition, p. 579

"I'm a frayed knot"
A string going into a bar for the third time
An old `string` joke

C.1 The Class `string`

The standard C++ string class is imported into client programs using:

```
#include<string>
```

It's possible you'll be programming in C++ using an older compiler that doesn't support the standard class, or that you'll be using a different string class, that is, the class `apstring` that is part of the Advanced Placement Computer Science C++ classes. A class `tstring` is provided with this book as a replacement for the standard class. It is identical to the class `apstring` except that the constant identifying an illegal position is `tstring::npos` instead of the global constant `npos` used in `apstring`.

The standard class `string` is better than a simple encapsulation of the C-style string which is a zero-terminated array of characters. The class `string` is actually a `typedef` for a templated class. The template makes it possible to change more easily to an alphabet with more characters than can be represented by a `char` value. The type `char` typically limits an alphabet to 128 or 256 different characters. I won't discuss the templated class `basic_string`. See one of the more advanced books on C++ for details, such as [Str97]. I will outline some of the member functions that you may find useful in writing programs. For information on all the `string` functions consult the header file `<string>` or a C++ reference.

C.1.1 Basic Operations

Strings can be read, written, assigned, copied, and compared using relational operators. The relational operators compare using **lexicographical** order, which is alphabetical order except that the underlying character set's ordinal values are used. This means that in an ASCII environment the string `"Zebra"` comes *before* the string `"aardvark"` because the ASCII value of the character ‘Z’ is 90 while the value of ‘a’ is 97.

Some string implementations may use efficient implementation techniques such as reference counting to share storage, but you can think of strings as working like the built-in types: assignment works as you should expect it to.

```
string a = "hello";
string b = a;        // b constructed as copy of a
b[0] = 'j';          // a still represents "hello"
```

As this example shows, individual characters are accessed using the indexing bracket `operator []`. There is no range-checking; an index that is greater than `s.length()-1`, the largest valid index, or less than zero, the smallest valid index, will be processed silently and almost certainly lead to an error later. The class `tstring`, like `apstring`, does do range-checking for the indexing operator. The standard class supports `at` which does do range-checking:

```
string s = "hello";
s[30] = 'x';      // problem eventually, bad index
s.at(30) = 'x';   // error, exception thrown
```

Characters are indexing beginning with zero, the last valid index is `s.length()-1` as we’ve noted. The function `length` returns the number of characters in the string which is one larger than the largest valid index because the first character has index zero.

C.1.2 Conversion to/from C-style Strings

Many C++ functions pre-date the C++ standard; other functions are written to be used with C-style strings. The method `string::c_str()` returns a C-style string equivalent to a string. We use this method extensively in opening text files.

```
string filename = "c:\\data\\hamlet.txt";
ifstream input(filename.c_str());         // open file
```

A string can also be constructed from a C-style string. This is how strings are constructed from string literals since a string literal is treated as a C-style string.

```
string s = "hello world"; // construct string(const char *)
```

Some of the useful C-style functions such as `atoi` and `atof` have equivalents in the library of string free functions from *strutils.h*, Program G.8. See How to G for details.

C.2 String Member Functions

C.2.1 Adding Characters or Strings

In this book we use overloaded `operator +=` and `operator +` extensively for appending characters to a string and concatenating strings, respectively.

```
string c = 'a';    // no good, cannot construct from char
string s = "hello";
string t = " world";
string u = "el";
u += "ephant";     // ok, u = "elephant"
u += 's';          // ok to append char, u = "elephants"
string v = s + t;  // ok, v = "hello world"
v = 't' + s;       // no good, can't concatenate to a char
v = string("") + 't' + s; // ok, join char to string
```

As shown, it's not possible to concatenate a string to a char, but it is possible to concatenate a char to a string. To guard against errors, there is no string constructor from one char, this is why concatenation of strings to chars doesn't work. It is possible, however, to concatenate a char to a string as shown in the examples above.

It's also possible to add a string (or a string literal/C-style string) at a given position/index using the method `string::insert`. The local copy below is needed because the parameters are `const`.

```
string Fullname(const string& first, const string& last)
// post: returns fullname, e.g., first + last
{
   // return first + " " + last;
   string copy(last);
   copy.insert(0," ");    // copy is now " " + last
   copy.insert(0,first);  // copy is now first + " " + last
   return copy;
}
```

C.2.2 Using Substrings

A substring can be extracted from a string using the method `string::substr()`. Substrings are specified using a starting index/position and the number of characters in the substring. The number of characters is optional.

```
string string::substr(int index = 0, int n = npos) const;
// post: return substring of n characters starting at index
```

The method `substr` "does the right thing" when too many characters are specified—only as many as are available are returned. On the other hand, if the starting position is out of range an error occurs. Program C.1 shows the `substr` method used together with other string methods we discuss in the next section. The function prototype above shows default values for both parameters, but the first argument is almost always provided.

The method `string::replace`, also shown in Program C.1, uses a position and a length to replace a substring of characters in a string.

```
string& string::replace(int index, int n, const string& s);
// post: replace n chars beginning at index with s,
//       return result
```

Thus `substr` reads (a copy of) part of a string and `replace` writes a part of a string. Both functions use as many characters as specified by the optional second parameter, but don't generate an error if there are fewer characters in the string than specified.

Program C.1 stringdemo.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s = "I sing the body electric";

    cout << s << endl;
    cout << s.substr(2,4) << endl;
    cout << s.substr(s.find("electric")–5) << endl << endl;

    string copy(s);
    int bodyPos = copy.find("body");
    copy.replace(bodyPos,copy.length(),"blues");
    cout << copy << endl << endl;

    cout << "search for chars/strings" << endl;
    cout << "first e at " << s.find('e')  << endl;
    cout << "last e at "  << s.rfind('e') << endl;
    cout << "first z at " << s.find('z')  << endl;
    cout << "space after body "   << s.find(" ", bodyPos) << endl;
    return 0;
}
```

stringdemo.cpp

OUTPUT

```
prompt> stringdemo
I sing the body electric
sing
body electric

I sing the blues

search for chars
first e at 9
last e at 18
first z at 4294967295
space after body at 15
```

C.2.3 Finding (Sub)strings and Characters

The string member functions `find` and `rfind` return the index in a string at which another string or character begins. If the searched-for value isn't found, the functions return `string::npos`. As the output shows, this is the largest positive value for an index. Your programs should not rely on `string::npos` having any particular value.

The functions `find` and `rfind` come in many flavors. We'll only use the basic versions though these have an optional second argument indicating at what index the search begins as shown in Program C.1.

```
int string::find(const string& s, int loc = 0) const;
// post: return position/index of first location of s
//       starting search at index loc,
//       return npos if not found

int string::find(char ch, int loc = 0) const;
// post: as above, search for character ch

int string::rfind(const string& s, int loc = 0) const;
// post: return position/index of first location of s
//       starting search at index loc, searching backwards
//       return npos if not found

int string::rfind(char ch, int loc = 0) const;
// post: as above, search for character ch backwards
```

Although we've used the type `int` for all indexes, the type `size_type` is actually used in all `string` member functions. In nearly every implementation this will be the same as `size_t`, an `unsigned int` or some other unsigned integer type, such as `long`.

Bibliography

[AA85] Donald J. Albers and G.L. Alexanderson. *Mathematical People*. Birkhäuser, 1985.

[ACM87] ACM. *Turing Award Lectures: The First Twenty Years 1966–1985*. ACM Press, 1987.

[AS96] Harold Abelson and Gerald Jay Sussman. *Structure and Interpretation of Computer Programs*. 2nd ed. MIT Press and McGraw-Hill, 1996.

[Asp90] William Aspray. *Computing Before Computers*. Iowa State University Press, 1990.

[Aus98] Matthew H. Austern *Generic Programming and the STL*. Addison-Wesley, 1998.

[Ben86] Jon Bentley. *Programming Pearls*. Addison-Wesley, 1986.

[Ben88] Jon Bentley. *More Programming Pearls*. Addison-Wesley, 1988.

[Ble90] Guy E. Blelloch. *Vector Models for Data-Parallel Computing*. MIT Press, 1990.

[Boo91] Grady Booch. *Object Oriented Design with Applications*. Benjamin Cummings, 1991.

[Boo94] Grady Booch. *Object Oriented Design and Analysis with Applications*. 2nd ed. Benjamin Cummings, 1994.

[BRE71] I. Barrodale, F.D. Roberts, and B.L. Ehle. *Elementary Computer Applications in Science Engineering and Business*. John Wiley & Sons Inc., 1971.

[Coo87] Doug Cooper. *Condensed Pascal*. W.W. Norton, 1987.

[Dij82] Edsger W. Dijkstra. *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982.

[DR90] Nachum Dershowitz and Edward M. Reingold. Calendrical calculations. *Software-Practice and Experience*, 20(9):899–928, September 1990.

[(ed91] Allen B. Tucker (ed.). *Computing Curricula 1991 Report of the ACM/IEEE-CS Joint Curriculum Task Force*. ACM Press, 1991.

[EL94] Susan Epstein and Joanne Luciano, editors. *Grace Hopper Celebration of Women in Computing*. Computing Research Association, 1994. Hopper-Book@cra.org.

[Emm93] Michele Emmer, editor. *The Visual Mind: Art and Mathematics*. MIT Press, 1993.

[G$\ddot{9}$5] Denise W. Gürer. Pioneering women in computer science. *Communications of the ACM*, 38(1):45–54, January 1995.

[Gar95] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1995.

[GHJ95] Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides *Design Patterns: Elements of Reusable Object-Oriented Programming* Addison-Wesley, 1995

[Gol93] Herman H. Goldstine. *The Computer from Pascal to von Neumann*. Princeton University Press, 1993.

[Gri74] David Gries. On structured programming - a reply to smoliar. *Communications of the ACM*, 17(11):655–657, 1974.

[GS93] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, 1993.

[Har92] David Harel. *Algorithmics, The Spirit of Computing*. Addison-Wesley, second edition, 1992.

[Hoa89] C.A.R. Hoare. *Essays in Computing Science*. Prentice-Hall, 1989. (editor) C.B. Jones.

[Hod83] Andrew Hodges. *Alan Turing: The Enigma*. Simon & Schuster, 1983.

[Hor92] John Horgan. Claude e. shannon. *IEEE Spectrum*, April 1992.

[JW89] William Strunk Jr. and E.B. White. *The Elements of Style*. MacMillan Publishing Co., third edition, 1989.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 1 Fundamental Algorithms. Addison-Wesley, third edition, 1997.

[Knu98a] Donald E. Knuth. *The Art of Computer Programming*, volume 2 Seminumerical Algorithms. Addison-Wesley, third edition, 1998.

[Knu98b] Donald E. Knuth. *The Art of Computer Programming*, volume 3 Sorting and Searching. Addison-Wesley, third edition 1998.

[KR78] Brian W. Kernighan and Dennis Ritchie. *The C Programming Language*. Prentice-Hall, 1978.

[KR96] Samuel N. Kamin and Edward M. Reingold. *Programming with class: A C++ Introduction to Computer Science*. McGraw-Hill, 1996.

[Mac92] Norman Macrae. *John von Neumann*. Pantheon Books, 1992.

June 7, 1999 10:10 owltex Sheet number 29 Page number 823 magenta black

[McC79] Pamela McCorduck. *Machines Who Think*. W.H. Freeman and Company, 1979.

[McC93] Steve McConnell. *Code Complete*. Microsoft Press, 1993.

[MGRS91] Albert R. Meyer, John V. Gutag, Ronald L. Rivest, and Peter Szolovits, editors. *Research Directions in Computer Science: An MIT Perspective*. MIT Press, 1991.

[Neu95] Peter G. Neumann. *Computer Related Risks*. Addison Wesley, 1995.

[Pat96] Richard E. Pattis. *Get A-Life: Advice for the Beginning Object-Oriented Programmer*. Turing TarPit Press, 1999.

[Per87] Alan Perlis. The synthesis of algorithmic systems. In *ACM Turing Award Lectures: The First Twenty Years*. ACM Press, 1987.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[RDC93] Edward M. Reingold, Nachum Dershowitz, and Stewart M. Clamen. Calendrical calculations, ii: Three historical calendars. *Software-Practice and Experience*, 23(4):383–404, April 1993.

[Rie96] Arthur Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.

[Rob95] Eric S. Roberts. Loop exits and structured programming: Reopening the debate. In *Papers of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, pages 268–272. ACM Press, March 1995. SIGCSE Bulletin V. 27 N 1.

[Rob95] Eric S. Roberts. *The Art and Science of C*. Addison-Wesley, 1995.

[Sla87] Robert Slater. *Portraits in Silicon*. MIT Press, 1987.

[Str87] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, 1987.

[Str94] Bjarne Stroustrup. *The Design and Evolution of C++*. Addison-Wesley, 1994.

[Str97] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, third edition, 1997.

[Mey92] Scott Meyers. *Effective C++*. Addison Wesley, 1992.

[Mey96] Scott Meyers. *More Effective C++*. Addison-Wesley, 1996.

[Wei94] Mark Allen Weiss. *Data Structures and Algorithm Analysis in C++*. Benjamin Cummings, 1994.

[Wil56] M.V. Wilkes. *Automatic Digital Computers*. John Wiley & Sons, Inc., 1956.

[Wil87] Maurice V. Wilkes. Computers then and now. In *ACM Turing Award Lectures: The First Twenty Years*, pages 197–205. ACM Press, 1987.

[Wil95] Maurice V. Wilkes. *Computing Perspectives*. Morgan Kaufmann, 1995.

[Wir87] Niklaus Wirth. From programming language design to compiler construction. In *ACM Turing Award Lectures: The First Twenty Years*. ACM Press, 1987.