



Chapter 9

Strings

Learning Objectives

- An Array Type for Strings
 - C-Strings
- Character Manipulation Tools
 - Character I/O
 - get, put member functions
 - putback, peek, ignore
- Standard Class string
 - String processing

Introduction

- Two string types:
- C-strings
 - Array with base type char
 - End of string marked with null, "\0"
 - "Older" method inherited from C
- String class
 - Uses templates

C-Strings

- Array with base type *char*
 - One character per indexed variable
 - One extra character: "\0"
 - Called "null character"
 - End marker
- We've used c-strings
 - Literal "Hello" stored as c-string

C-String Variable

- Array of characters:
char s[10];
 - Declares a c-string variable to hold up to 9 characters
 - + one null character
- Typically "partially-filled" array
 - Declare large enough to hold max-size string
 - Indicate end with null
- Only difference from standard array:
 - Must contain null character

C-String Storage

- A standard array:

```
char s[10];
```

- If s contains string "Hi Mom", stored as:

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	i		M	o	m	!	\0	?	?

C-String Initialization

- Can initialize c-string:

```
char myMessage[20] = "Hi there.";
```

- Needn't fill entire array
- Initialization places "\0" at end

- Can omit array-size:

```
char shortString[] = "abc";
```

- Automatically makes size one more than length of quoted string
- NOT same as:

```
char shortString[] = {"a", "b", "c"};
```

C-String Indexes

- A c-string IS an array
- Can access indexed variables of:
`char ourString[5] = "Hi";`
 - ourString[0] is "H"
 - ourString[1] is "i"
 - ourString[2] is "\0"
 - ourString[3] is unknown
 - ourString[4] is unknown

C-String Index Manipulation

- Can manipulate indexed variables

```
char happyString[7] = "DoBeDo";  
happyString[6] = "Z";
```

- Be careful!
 - Here, "\0" (null) was overwritten by a "Z"!
- If null overwritten, c-string no longer "acts" like c-string!
 - Unpredictable results!

Library

- Declaring c-strings
 - Requires no C++ library
 - Built into standard C++
- Manipulations
 - Require library `<cstring>`
 - Typically included when using c-strings
 - Normally want to do "fun" things with them

= and == with C-strings

- C-strings not like other variables
 - Cannot assign or compare:

```
char aString[10];  
aString = "Hello"; // ILLEGAL!
```

 - Can ONLY use "=" at declaration of c-string!
- Must use library function for assignment:

```
strcpy(aString, "Hello");
```

 - Built-in function (in <cstring>)
 - Sets value of aString equal to "Hello"
 - NO checks for size!
 - Up to programmer, just like other arrays!

Comparing C-strings

- Also cannot use operator ==

```
char aString[10] = "Hello";  
char anotherString[10] = "Goodbye";  
– aString == anotherString;    // NOT allowed!
```

- Must use library function again:

```
if (strcmp(aString, anotherString))  
    cout << "Strings NOT same."  
else  
    cout << "Strings are same."
```

The <cstring> Library:

Display 9.1 Some Predefined C-String Functions in <cstring> (1 of 2)

- Full of string manipulation functions

Display 9.1 Some Predefined C-String Functions in <cstring>

FUNCTION	DESCRIPTION	CAUTIONS
<code>strcpy(Target_String_Var, Src_String)</code>	Copies the C-string value <i>Src_String</i> into the C-string variable <i>Target_String_Var</i> .	Does not check to make sure <i>Target_String_Var</i> is large enough to hold the value <i>Src_String</i> .
<code>strncpy(Target_String_Var, Src_String, Limit)</code>	The same as the two-argument <code>strcpy</code> except that at most <i>Limit</i> characters are copied.	If <i>Limit</i> is chosen carefully, this is safer than the two-argument version of <code>strcpy</code> . Not implemented in all versions of C++.
<code>strcat(Target_String_Var, Src_String)</code>	Concatenates the C-string value <i>Src_String</i> onto the end of the C-string in the C-string variable <i>Target_String_Var</i> .	Does not check to see that <i>Target_String_Var</i> is large enough to hold the result of the concatenation.

(continued)

The <cstring> Library:

Display 9.1 Some Predefined C-String Functions in <cstring> (2 of 2)

Display 9.1 Some Predefined C-String Functions in <cstring>

FUNCTION	DESCRIPTION	CAUTIONS
<code>strcat(<i>Target_String_Var</i>, <i>Src_String</i>, <i>Limit</i>)</code>	The same as the two argument <code>strcat</code> except that at most <i>Limit</i> characters are appended.	If <i>Limit</i> is chosen carefully, this is safer than the two-argument version of <code>strcat</code> . Not implemented in all versions of C++.
<code>strlen(<i>Src_String</i>)</code>	Returns an integer equal to the length of <i>Src_String</i> . (The null character, ' <code>\0</code> ', is not counted in the length.)	
<code>strcmp(<i>String_1</i>, <i>String_2</i>)</code>	Returns 0 if <i>String_1</i> and <i>String_2</i> are the same. Returns a value < 0 if <i>String_1</i> is less than <i>String_2</i> . Returns a value > 0 if <i>String_1</i> is greater than <i>String_2</i> (that is, returns a nonzero value if <i>String_1</i> and <i>String_2</i> are different). The order is lexicographic.	If <i>String_1</i> equals <i>String_2</i> , this function returns 0, which converts to <code>false</code> . Note that this is the reverse of what you might expect it to return when the strings are equal.
<code>strcmp(<i>String_1</i>, <i>String_2</i>, <i>Limit</i>)</code>	The same as the two-argument <code>strcat</code> except that at most <i>Limit</i> characters are compared.	If <i>Limit</i> is chosen carefully, this is safer than the two-argument version of <code>strcmp</code> . Not implemented in all versions of C++.

C-string Functions: strlen()

- "String length"
- Often useful to know string length:

```
char myString[10] = "dobedo";  
cout << strlen(myString);
```

- Returns number of characters

- Not including null

- Result here:

6

C-string Functions: strcat()

- `strcat()`

- "String concatenate":

```
char stringVar[20] = "The rain";  
strcat(stringVar, "in Spain");
```

- Note result:

- stringVar now contains "The rainin Spain"

- Be careful!

- Incorporate spaces as needed!

C-string Arguments and Parameters

- Recall: c-string is array
- So c-string parameter is array parameter
 - C-strings passed to functions can be changed by receiving function!
- Like all arrays, typical to send size as well
 - Function "could" also use "\0" to find end
 - So size not necessary if function won't change c-string parameter
 - Use "const" modifier to protect c-string arguments

C-String Output

- Can output with insertion operator, <<
- As we've been doing already:
`cout << news << " Wow.\n";`
 - Where *news* is a c-string variable
- Possible because << operator is overloaded for c-strings!

C-String Input

- Can input with extraction operator, >>
 - Issues exist, however
- Whitespace is "delimiter"
 - Tab, space, line breaks are "skipped"
 - Input reading "stops" at delimiter
- Watch size of c-string
 - Must be large enough to hold entered string!
 - C++ gives no warnings of such issues!

C-String Input Example

- ```
char a[80], b[80];
cout << "Enter input: ";
cin >> a >> b;
cout << a << b << "END OF OUTPUT\n";
```
- Dialogue offered:  
Enter input: Do be do to you!  
DobeEND OF OUTPUT  
– Note: Underlined portion typed at keyboard
- C-string *a* receives: "do"
- C-string *b* receives: "be"

# C-String Line Input

- Can receive entire line into c-string
- Use `getline()`, a predefined member function:

```
char a[80];
cout << "Enter input: ";
cin.getline(a, 80);
cout << a << "END OF OUTPUT\n";
```

– Dialogue:

Enter input: Do be do to you!

Do be do to you!END OF INPUT

# Example: Command Line Arguments

- Programs invoked from the command line (e.g. a UNIX shell, DOS command prompt) can be sent arguments
  - Example: `COPY C:\FOO.TXT D:\FOO2.TXT`
    - This runs the program named “COPY” and sends in two C-String parameters, “C:\FOO.TXT” and “D:\FOO2.TXT”
    - It is up to the COPY program to process the inputs presented to it; i.e. actually copy the files
- Arguments are passed as an array of C-Strings to the main function

# Example: Command Line Arguments

- Header for main
  - `int main(int argc, char *argv[])`
  - `argc` specifies how many arguments are supplied. The name of the program counts, so `argc` will be at least 1.
  - `argv` is an array of C-Strings.
    - `argv[0]` holds the name of the program that is invoked
    - `argv[1]` holds the name of the first parameter
    - `argv[2]` holds the name of the second parameter
    - Etc.

# Example: Command Line Arguments

```
// Echo back the input arguments
int main(int argc, char *argv[])
{
 for (int i=0; i<argc; i++)
 {
 cout << "Argument " << i << " " << argv[i] << endl;
 }
 return 0;
}
```

## Sample Execution

> Test  
Argument 0 Test

**Invoking Test  
from command  
prompt**

## Sample Execution

> Test hello world  
Argument 0 Test  
Argument 1 hello  
Argument 2 world



# More getline()

- Can explicitly tell length to receive:

```
char shortString[5];
cout << "Enter input: ";
cin.getline(shortString, 5);
cout << shortString << "END OF OUTPUT\n";
```

- Results:

Enter input: dobedowap  
dobeEND OF OUTPUT

- Forces FOUR characters only be read
    - Recall need for null character!

# Character I/O

- Input and output data
  - ALL treated as character data
  - e.g., number 10 outputted as "1" and "0"
  - Conversion done automatically
    - Uses low-level utilities
- Can use same low-level utilities ourselves as well

# Member Function get()

- Reads one char at a time
- Member function of cin object:  
`char nextSymbol;`  
`cin.get(nextSymbol);`
  - Reads next char & puts in variable nextSymbol
  - Argument must be char type
    - Not "string"!

# Member Function put()

- Outputs one character at a time
- Member function of cout object:
- Examples:

```
cout.put("a");
```

- Outputs letter "a" to screen

```
char myString[10] = "Hello";
cout.put(myString[1]);
```

- Outputs letter "e" to screen

# More Member Functions

- `putback()`
  - Once read, might need to "put back"
  - `cin.putback(lastChar);`
- `peek()`
  - Returns next char, but leaves it there
  - `peekChar = cin.peek();`
- `ignore()`
  - Skip input, up to designated character
  - `cin.ignore(1000, "\n");`
    - Skips at most 1000 characters until "\n"

# Character-Manipulating Functions:

## Display 9.3 Some Functions in <cctype> (1 of 3)

**Display 9.3** Some Functions in <cctype>

| FUNCTION                       | DESCRIPTION                                                                              | EXAMPLE                                                                                                             |
|--------------------------------|------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <code>toupper(Char_Exp)</code> | Returns the uppercase version of <i>Char_Exp</i> (as a value of type <code>int</code> ). | <pre>char c = toupper('a');<br/>cout &lt;&lt; c;<br/>Outputs: A</pre>                                               |
| <code>tolower(Char_Exp)</code> | Returns the lowercase version of <i>Char_Exp</i> (as a value of type <code>int</code> ). | <pre>char c = tolower('A');<br/>cout &lt;&lt; c;<br/>Outputs: a</pre>                                               |
| <code>isupper(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is an uppercase letter; otherwise, returns false.  | <pre>if (isupper(c))<br/>    cout &lt;&lt; "Is uppercase."<br/>else<br/>    cout &lt;&lt; "Is not uppercase."</pre> |

# Character-Manipulating Functions:

## Display 9.3 Some Functions in <cctype> (2 of 3)

**Display 9.3** Some Functions in <cctype>

| FUNCTION                       | DESCRIPTION                                                                                           | EXAMPLE                                                                                                                                                                        |
|--------------------------------|-------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>islower(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is a lowercase letter; otherwise, returns false.                | <pre>char c = 'a'; if (islower(c))     cout &lt;&lt; c &lt;&lt; " is lowercase."; <b>Outputs:</b> a is lowercase.</pre>                                                        |
| <code>isalpha(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is a letter of the alphabet; otherwise, returns false.          | <pre>char c = '\$'; if (isalpha(c))     cout &lt;&lt; "Is a letter."; else     cout &lt;&lt; "Is not a letter."; <b>Outputs:</b> Is not a letter.</pre>                        |
| <code>isdigit(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is one of the digits '0' through '9'; otherwise, returns false. | <pre>if (isdigit('3'))     cout &lt;&lt; "It's a digit."; else     cout &lt;&lt; "It's not a digit."; <b>Outputs:</b> It's a digit.</pre>                                      |
| <code>isalnum(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is either a letter or a digit; otherwise, returns false.        | <pre>if (isalnum('3') &amp;&amp; isalnum('a'))     cout &lt;&lt; "Both alphanumeric."; else     cout &lt;&lt; "One or more are not."; <b>Outputs:</b> Both alphanumeric.</pre> |

# Character-Manipulating Functions:

## Display 9.3 Some Functions in <cctype> (3 of 3)

|                                |                                                                                                                                      |                                                                                                                                                          |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>isspace(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is a whitespace character, such as the blank or newline character; otherwise, returns false.   | <pre>//Skips over one "word" and sets c //equal to the first whitespace //character after the "word": do {     cin.get(c); } while (! isspace(c));</pre> |
| <code>ispunct(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is a printing character other than whitespace, a digit, or a letter; otherwise, returns false. | <pre>if (ispunct('?'))     cout &lt;&lt; "Is punctuation."; else     cout &lt;&lt; "Not punctuation.";</pre>                                             |
| <code>isprint(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is a printing character; otherwise, returns false.                                             |                                                                                                                                                          |
| <code>isgraph(Char_Exp)</code> | Returns true provided <i>Char_Exp</i> is a printing character other than whitespace; otherwise, returns false.                       |                                                                                                                                                          |
| <code>isctrl(Char_Exp)</code>  | Returns true provided <i>Char_Exp</i> is a control character; otherwise, returns false.                                              |                                                                                                                                                          |



# Standard Class string

- Defined in library:

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

- String variables and expressions
  - Treated much like simple types

- Can assign, compare, add:

```
string s1, s2, s3;
s3 = s1 + s2; //Concatenation
s3 = "Hello Mom!" //Assignment
```

- Note c-string "Hello Mom!" automatically converted to string type!

# Display 9.4

## Program Using the Class string

**Display 9.4** Program Using the Class string

```
1 //Demonstrates the standard class string.
2 #include <iostream>
3 #include <string>
4 using namespace std;

5 int main()
6 {
7 string phrase;
8 string adjective("fried"), noun("ants");
9 string wish = "Bon appetite!";

10 phrase = "I love " + adjective + " " + noun + "!";
11 cout << phrase << endl
12 << wish << endl;

13 return 0;
14 }
```

*Initialized to the empty string.*

*Two equivalent ways of initializing a string variable*

### SAMPLE DIALOGUE

I love fried ants!  
Bon appetite!

# I/O with Class string

- Just like other types!
- ```
string s1, s2;  
cin >> s1;  
cin >> s2;
```
- Results:
User types in:
May the hair on your toes grow long and curly!
- Extraction still ignores whitespace:
s1 receives value "May"
s2 receives value "the"

getline() with Class string

- For complete lines:

```
string line;  
cout << "Enter a line of input: ";  
getline(cin, line);  
cout << line << "END OF OUTPUT";
```

- Dialogue produced:

Enter a line of input: Do be do to you!

Do be do to you!END OF INPUT

– Similar to c-string's usage of getline()

Other getline() Versions

- Can specify "delimiter" character:

```
string line;  
cout << "Enter input: ";  
getline(cin, line, "?");
```

- Receives input until "?" encountered

- getline() actually returns reference

- ```
string s1, s2;
getline(cin, s1) >> s2;
```

- Results in: 

```
(cin) >> s2;
```

# Pitfall: Mixing Input Methods

- Be careful mixing `cin >> var` and `getline`

```
– int n;
 string line;
 cin >> n;
 getline(cin, line);
```

– If input is:           42  
                          Hello hitchhiker.

- Variable `n` set to 42
- `line` set to empty string!

– `cin >> n` skipped leading whitespace, leaving `"\n"` on stream for `getline()`!

# Class string Processing

- Same operations available as c-strings
- And more!
  - Over 100 members of standard string class
- Some member functions:
  - `.length()`
    - Returns length of string variable
  - `.at(i)`
    - Returns reference to char at position i

# Display 9.7 Member Functions of the Standard Class string (1 of 2)

**Display 9.7**    **Member Functions of the Standard Class string**

| EXAMPLE                                   | REMARKS                                                                                                                                                                              |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Constructors</b>                       |                                                                                                                                                                                      |
| <code>string str;</code>                  | Default constructor; creates empty string object <code>str</code> .                                                                                                                  |
| <code>string str("string");</code>        | Creates a string object with data "string".                                                                                                                                          |
| <code>string str(aString);</code>         | Creates a string object <code>str</code> that is a copy of <code>aString</code> . <code>aString</code> is an object of the class <code>string</code> .                               |
| <b>Element access</b>                     |                                                                                                                                                                                      |
| <code>str[i]</code>                       | Returns read/write reference to character in <code>str</code> at index <code>i</code> .                                                                                              |
| <code>str.at(i)</code>                    | Returns read/write reference to character in <code>str</code> at index <code>i</code> .                                                                                              |
| <code>str.substr(position, length)</code> | Returns the substring of the calling object starting at position and having <code>length</code> characters.                                                                          |
| <b>Assignment/Modifiers</b>               |                                                                                                                                                                                      |
| <code>str1 = str2;</code>                 | Allocates space and initializes it to <code>str2</code> 's data, releases memory allocated for <code>str1</code> , and sets <code>str1</code> 's size to that of <code>str2</code> . |
| <code>str1 += str2;</code>                | Character data of <code>str2</code> is concatenated to the end of <code>str1</code> ; the size is set appropriately.                                                                 |
| <code>str.empty( )</code>                 | Returns true if <code>str</code> is an empty string; returns false otherwise.                                                                                                        |

(continued)



# Display 9.7 Member Functions of the Standard Class string (2 of 2)

**Display 9.7**    **Member Functions of the Standard Class string**

---

| EXAMPLE                                                   | REMARKS                                                                                                                                                       |
|-----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>str1 + str2</code>                                  | Returns a string that has <code>str2</code> 's data concatenated to the end of <code>str1</code> 's data. The size is set appropriately.                      |
| <code>str.insert(pos, str2)</code>                        | Inserts <code>str2</code> into <code>str</code> beginning at position <code>pos</code> .                                                                      |
| <code>str.remove(pos, length)</code>                      | Removes substring of size <code>length</code> , starting at position <code>pos</code> .                                                                       |
| <b>Comparisons</b>                                        |                                                                                                                                                               |
| <code>str1 == str2</code> <code>str1 != str2</code>       | Compare for equality or inequality; returns a Boolean value.                                                                                                  |
| <code>str1 &lt; str2</code> <code>str1 &gt; str2</code>   | Four comparisons. All are lexicographical comparisons.                                                                                                        |
| <code>str1 &lt;= str2</code> <code>str1 &gt;= str2</code> |                                                                                                                                                               |
| <code>str.find(str1)</code>                               | Returns index of the first occurrence of <code>str1</code> in <code>str</code> .                                                                              |
| <code>str.find(str1, pos)</code>                          | Returns index of the first occurrence of string <code>str1</code> in <code>str</code> ; the search starts at position <code>pos</code> .                      |
| <code>str.find_first_of(str1, pos)</code>                 | Returns the index of the first instance in <code>str</code> of any character in <code>str1</code> , starting the search at position <code>pos</code> .        |
| <code>str.find_first_not_of(str1, pos)</code>             | Returns the index of the first instance in <code>str</code> of any character <i>not</i> in <code>str1</code> , starting search at position <code>pos</code> . |

# C-string and string Object Conversions

- Automatic type conversions
  - From c-string to string object:

```
char aCString[] = "My C-string";
string stringVar;
stringVar = aCString;
```

    - Perfectly legal and appropriate!
  - `aCString = stringVar;`
    - ILLEGAL!
    - Cannot auto-convert to c-string
  - Must use explicit conversion:

```
strcpy(aCString, stringVar.c_str());
```

# Converting between `string` and numbers

- In C++11 it is simply a matter of calling **`stof`**, **`stod`**, **`stoi`**, or **`stol`** to convert a string to a float, double, int, or long, respectively.

```
int i;
double d;
string s;
i = stoi("35"); // Converts the string "35" to an integer 35
d = stod("2.5"); // Converts the string "2.5" to the double 2.5
```

# Converting between numbers and `string` objects

- In C++11 use **`to_string`** to convert a numeric type to a string

```
string s;
s = to_string(d*2); // Converts the double 5.0 to a
 // string "5.0000"
```

# Summary

- C-string variable is "array of characters"
  - With addition of null character, "\0"
- C-strings act like arrays
  - Cannot assign, compare like simple variables
- Libraries <cctype> & <string> have useful manipulating functions
- cin.get() reads next single character
- getline() versions allow full line reading
- Class string objects are better-behaved than c-strings