The University Of Michigan

EECS402 Lecture 08

Andrew M. Morgan

Savitch Ch. 9.0
C++ String Data Type
C-Strings

Intro To The Standard string Class

- C++ has a standard class called "string"
- Strings are simply a sequence of characters
  - Note: This is not a sufficient definition for a "C-string"
  - A "C-string" is an array of characters terminated by a null byte
    - More on this later…
- Must #include <string> using the standard namespace to get C++ standard string functionality
  - Note: This is different from #include'ing <string.h> which is the header required for "C-string"s
- string variables are used to store names, words, phrases, etc.
- Can be input using ">>" and output using "<<" as other types

Some string Functionality

- Declaring a string:
  - string lastName;
  - string firstName("Drew"); //Note: String literal enclosed in double quotes
  - string fullName;
- Assigning a string:
  - lastName = "Morgan"; //The usual assignment operator
- Appending one string on the end of another:
  - fullName = firstName + lastName; //Results in "DrewMorgan"
  - fullName = firstName + " " + lastName; //Results in "Drew Morgan"
- Accessing individual characters in a string:
  - myChar = firstName[2]; //Results in 'e' (no bounds checking)
  - myChar = firstName.at(2); //Results in 'e' (does bounds checking)
- Appending a character to the end of a string:
  - lastName = lastName + myChar; //Results in "Morgane"
- Determining number of characters in string:
  - myInt = firstName.length(); //Results in 4

string Example #1


```
#include <iostream>
#include <string>
using namespace std;
int main(void)
{
  string first;
  string last("Morgan");

  first = "Drew"; //Would be illegal for C-string
  cout << "Length of " << first << " is: " << first.length() << endl;
  cout << "Length of " << last << " is: " << last.length() << endl;

  first += "Morgan";
  cout << "Length of " << first << " is: " << first.length() << endl;
  cout << "Length of " << last << " is: " << last.length() << endl;

  first.assign("Drew");
  first.append(" ");
  first.append(last);
  cout << "Length of " << first << " is: " << first.length() << endl;
  cout << "Length of " << last << " is: " << last.length() << endl;
  return 0;
}
```

```
Length of Drew is: 4
Length of Morgan is: 6
Length of DrewMorgan is: 10
Length of Morgan is: 6
Length of Drew Morgan is: 11
Length of Morgan is: 6
```

Additional string Functionality

- Strings can be compared with usual operators
  - >, >= (greater than, greater than/equal to)
  - <, <= (less than, less than/equal to)
  - == (equality)
- Strings also have a member function called "compare"
  - int string::compare(string rhs);
  - Return value is negative if calling string is less than rhs
  - Return value is positive if calling string is greater than rhs
  - Return value is zero if both strings are identical

string Example #2


```
int main(void)
{
  string s1 = "Drew";
  string s3;
  int result;

  s3 = "Bob";
  if (s3 < s1)
    cout << "oper: s3 less than s1";
  if (s3 > s1)
    cout << "oper: s3 greater than s1";
  if (s3 == s1)
    cout << "oper: s3 is equal to s1";
  cout << endl;

  result = s3.compare(s1);
  if (result < 0)
    cout << "comp: s3 less than s1";
  else if (result > 0)
    cout << "comp: s3 greater than s1";
  else
    cout << "comp: s3 is equal to s1";
  cout << endl;

  s3 = "Drew";
  if (s3 < s1)
    cout << "oper: s3 less than s1";
  if (s3 > s1)
    cout << "oper: s3 greater than s1";
  if (s3 == s1)
    cout << "oper: s3 is equal to s1";
  cout << endl;

  result = s3.compare(s1);
  if (result < 0)
    cout << "comp: s3 less than s1";
  else if (result > 0)
    cout << "comp: s3 greater than s1";
  else
    cout << "comp: s3 is equal to s1";
  cout << endl;

  return 0;
}
```

```
oper: s3 less than s1
comp: s3 less than s1
oper: s3 is equal to s1
comp: s3 is equal to s1
```

Even More string Functionality

- Getting a substring of a string:
  - string string::substr(int startPos, int length)
    - Returns the substring starting at "startPos" with length of "length"
- Finding the location of a substring within a string:
  - int string::find(string lookFor);
    - Returns the index where the first instance of "lookFor" was found in the string
    - Returns "string::npos" (which is usually -1) when the substring isn't found
  - int string::find(string lookFor, int startFrom);
    - Returns the index where the first instance of "lookFor" was found, starting the search at the index "startFrom", or "string::npos" when the substring isn't found
- Finding specific characters in a string:
  - int string::find_first_of(string charList, int startFrom);
    - Returns the index of the first instance of any character in "charList", starting the search at the index "startFrom", or "string::npos" if none of the chars are found
  - int string::find_first_not_of(string charList, int startFrom);
    - Returns the index of the first instance of any character NOT in "charList", starting the search at the index "startFrom", or "string::npos" if none of the chars are found

A set of characters, see example b

string Example #3


```
int main()
{
  int startPos;
  int len;
  int commaLoc;
  int howLoc;
  int loc;
  int spaceLoc;
  string myStr;
  string myStr2;

  myStr = "Hello, how are you?";
  startPos = 7;
  len = 3;
  myStr2 = myStr.substr(startPos, len);
  cout << "Substr: " << myStr2 << endl;
  commaLoc = myStr.find(",");
  howLoc = myStr.find(myStr2);
  cout << "Comma: " << commaLoc;
  cout << " how: " << howLoc << endl;
  cout << "Spaces:";
  spaceLoc = myStr.find(" ");
  while (spaceLoc != string::npos)
  {
    cout << " " << spaceLoc;
    spaceLoc = myStr.find(" ", spaceLoc + 1);
  }
  cout << endl;

  cout << "Punct and spaces:";
  loc = myStr.find_first_of(" ,?", 0);
  while (loc != string::npos)
  {
    cout << " " << loc;
    loc = myStr.find_first_of(" ,?", loc + 1);
  }
  cout << endl;

  return 0;
}
```

```
Substr: how
Comma: 5 how: 7
Spaces: 6 10 14
Punct and spaces: 5 6 10 14 18
```

string Class Implementation

- The string class uses dynamic memory allocation to be sure segmentation faults don't occur
  - When a string is updated such that it requires more characters than currently allocated, a new, larger array is allocated and the prior contents are copied over as necessary
- Since dynamic allocation is relatively slow, it is not desirable to be re-allocating strings often
  - C++ allows some memory to be "wasted" by often allocating more space than is really needed
  - However, as strings are appended to the end, it is likely that a re-allocation won't be needed every time
  - Occasionally, re-allocation is necessary and is performed, again allocating more memory than necessary
- Note: this is all done *automatically* by the string class

Some Final string Functionality

- Several member functions are available to get information about a string
  - capacity: The number of characters that can be placed in a string without the inefficiency of re-allocating
  - length: The number of characters currently in the string
- You can manually change the capacity of a string
  - resize: Sets the capacity of a string to be at least a user-defined size
  - This can be useful if you know a string will be at most *n* characters long
    - By resizing the string to capacity *n* only that amount of memory is associated with the string
    - This prevents wasted memory when you know the exact size you need
    - Additionally, it can help prevent numerous re-allocations if you will be appending on to the end of the string, but know the final size ahead of time

string Example #4


```
int main(void)
{
  string str;
  string str2;

  cout << "Str: " << str << endl;
  cout << "Length: " << str.length();
  cout << "   Cap: " << str.capacity();
  cout << endl;

  str = "888";
  cout << "Str: " << str << endl;
  cout << "Length: " << str.length();
  cout << "   Cap: " << str.capacity();
  cout << endl;

  str += "-111-";
  cout << "Str: " << str << endl;
  cout << "Length: " << str.length();
  cout << "   Cap: " << str.capacity();
  cout << endl;

  str += "1723-9";
  cout << "Str: " << str << endl;
  cout << "Length: " << str.length();
  cout << "   Cap: " << str.capacity();
  cout << endl;

  str += "abcdefghijklmnopqrstuv";
  cout << "Str: " << str << endl;
  cout << "Length: " << str.length();
  cout << "   Cap: " << str.capacity();
  cout << endl;

  return 0;
}
```

```
Str:
Length: 0   Cap: 0
Str: 888
Length: 3   Cap: 31
Str: 888-111-
Length: 8   Cap: 31
Str: 888-111-1723-9
Length: 14   Cap: 31
Str: 888-111-1723-9abcdefghijklmnopqrstuv
Length: 36   Cap: 63
```


C-Strings: Arrays Of Characters

- Arrays of characters can be treated differently than arrays of other types
- Array of characters, terminated with a NULL: C-string
- Not all character arrays are strings
  - Must include the NULL character on end
  - Must ensure size of array allows room for NULL
- C-strings can be output directly using <<
  - Other arrays can not!
- There are many predefined functions in header file string.h for modifying strings that will be discussed
  - Note: <string.h> is for C-strings, <string> is for the C++ standard string

    or <cstring>

Simple C-string Program


```
int main(void)
{
  const int SIZE = 5;
  int i = 0;
  int iary[SIZE] = {2,4,6,8,10}; //NOT a C-string
  char cary[SIZE] = {'D','r','e','w','\0'}; //IS a C-string
  char cary2[SIZE] = "Drew"; //NULL automatic! - IS a C-string
  char cary3[SIZE] = {'H','e','l','l','o'}; //NOT a C-string


  cout << iary << endl;
  cout << cary << endl;
  cout << cary2 << endl;

  return 0;
}
```

0 works too

```
0xffbef8e0
Drew
Drew
```

These are POTENTIAL results. First line is an address of the first element of the iary. Your specific results will vary (printing a different address)

Common Problem

Initial Mem.

1000	t
1001	ÿ
1002	4
1003	ÿ
1004	ÿ
1005	¾
1006	ú
1007	L
1008	0
1009	R
100A	$

```
int main(void)
{
  const int SIZE = 4;
  char ary[SIZE]={'D','r','e','w'};

  cout << ary << endl;

  return 0;
}
```

Drewÿ¾úL

Resulting Mem.

1000	~~t~~	D
1001	~~ÿ~~	r
1002	~~4~~	e
1003	~~ÿ~~	w
1004	ÿ	
1005	¾	
1006	ú	
1007	L	
1008	0	
1009	R	
100A	$	

Note: NULL character (0)

Assigning C-strings

- While initialization and assignment seem the same, they are two different operations
- Assigning a C-string to a literal string (i.e. "Drew") is legal during initialization, using = at declaration-time
- Assigning a C-string to a literal string, using operator=, is NOT legal anywhere else in the program!
- C-string assignment can still be done, but you must call a function in string.h
- Function prototype:

```
char * strcpy(char *dest, char *src);
```


Using strcpy()


```
const int SIZE=5;
char cary[SIZE] = "Drew"; //Legal here!
char cary2[SIZE];

//cary2 = "Drew";   //ACK! Don't do this!

//strcpy() automatically appends a NULL
//character to the end of the string.
strcpy(cary2, "Drew"); //Ahh. Much better.

cout << cary << endl;
cout << cary2 << endl;
```

Drew
Drew

Comparing and Appending Two Strings

- Like assignment, comparison is not allowed with the "==" operator
  - Since these "strings" are really just character pointers, these operators would work on the pointer values (addresses), rather than the contents
- Must call a function from string.h. Prototype:

```
int strcmp(char *s1, char *s2);
```

  - s1 and s2 are C-strings (char arrays, terminated with NULL)
  - Return integer is:
    - 0 if strings are the same
    - negative if s1 is "less than" s2 (not the same)
    - positive is s1 is "greater than" s2 (not the same)
- Appending strings is done with a funtion. Prototype:

```
char * strcat(char *s1, char *s2);
```

  - s1 and s2 are C-strings (char arrays, terminated with NULL)
  - If s1 was "Drew" and s2 was "Morgan" then after a call to strcat, s1 would contain "DrewMorgan" and s2 is unchanged

Using strcmp() And strcat()


```
const int SIZE=5;
char cary[SIZE] = "Drew";
char cary2[SIZE] = "Blah";

//Don't forget the "==0" part!!!
if (strcmp(cary, cary2) == 0)
  cout << "Same strings!" << endl;
else
  cout << "Not the same!" << endl;

//Don't forget the "==0" part!!!
if (strcmp(cary, "Drew") == 0)
  cout << "Same strings!" << endl;
else
  cout << "Not the same!" << endl;
```

Not the same!
Same strings!

```
const int SIZE=15;
char cary[SIZE] = "Drew"; //Need not fill array
char cary2[SIZE] = "Morgan";

cout << "Before: " << cary << endl;

strcat(cary, " ");
strcat(cary, cary2);

cout << "After: " << cary << endl;
```

Before: Drew
After: Drew Morgan

Finding The Length of a String

- You often want to know how long a string is. Prototype:

```
int strlen(char *s);
```

  - The int being returned is the length of the string
  - It is not the length of the array
  - It is not the length of the string including the NULL
- Example:

```
cary[15] = "Drew";
cout << "Length: " << strlen(cary) << endl;
```

Length: 4

Arrays of C-Strings

- C-Strings are just arrays, use 2-D array for an array of strings

```
char strAry[4][4] = {"and", "the", "one", "for"};
int i;
strAry[0][1] = 'b';
strAry[0][2] = 'c';
strcpy(strAry[2], "two");

for (i = 0; i < 4; i++)
{
  cout << strAry[i] << endl;
}
```

abc
the
two
for

Command Line Parameters

- Until now, main has taken no parameters
- Often you want to provide inputs to main, without requiring user interaction
- Main can optionally take two parameters
  - int argc: A count of the number of items typed on the command line
    - Name of executable counts too!
  - char *argv[]: An array of C-strings, one per item on the command line
    - Name of executable is *always* argv[0]
    - Each item is a C-string
      - Even if you type "147' it will be treated as the C-string "147", NOT the integer 147

Command Line Parameters - Example


```
int main(int argc, char *argv[])
{
  int i;

  for (i = 0; i < argc; i++)
  {
    cout << "Item #" << i << ": " << argv[i] << endl;
  }

  return 0;
}
```

```
cmdPrompt% ./cmdLineDemo
Item #0: ./cmdLineDemo
cmdPrompt% ./cmdLineDemo hello 147 88.3
Item #0: ./cmdLineDemo
Item #1: hello
Item #2: 147
Item #3: 88.3
```

Usage Statements In Programs

- If your program requires input parameters, it should include a usage statement
  - How the user is notified about the correct way to run your program

```
int main(int argc, char *argv[])
{
  ifstream inFile;

  if (argc != 3)
  {
    cout << "Usage: " << argv[0] << " <inputFilename> <value>" << endl;
    exit(0);
  }

  inFile.open(argv[1]);
  cout << "Value is: " << argv[2] << endl;

  //...  more code to read
  //... input file, etc...
  return 0;
}
```

```
cmdPrompt% usageDemo
Usage: usageDemo <inputFilename> <value>

cmdPrompt% usageDemo infile.txt
Usage: usageDemo <inputFilename> <value>

cmdPrompt% usageDemo infile.txt 45 extraInfo
Usage: usageDemo <inputFilename> <value>

cmdPrompt% usageDemo infile.txt 45
Value is: 45
```

This is a cstring

Simple Conversion of C-Strings

- There are some function that allow easy conversion of C-strings to numeric types
  - int atoi(char *cstring)
    - Converts the ASCII C-string "cstring" to its integer representation.
    - atoi is read "ASCII to Integer"
    - If conversion isn't possible (invalid format), return value is undefined (usually 0)
    - No good way to perform error checking, since any integer that it returns would be a valid integer
  - double atof(char *cstring)
    - Converts the ASCII C-string "cstring" to its double precision floating point representation.
    - atof is read "ASCII to Float" (actually returns a double)
    - If conversion isn't possible (invalid format), return value is undefined (usually 0)
    - No good way to perform error checking, since any integer that it returns would be a valid double

Simple Conversion of C-Strings, Example


```
int main(int argc, char *argv[])
{
  int intVal;
  double doubleVal;

  if (argc != 3)
  {
    cout << "Usage: " << argv[0] << " <floatVal> <intVal>" << endl;
    exit(2);
  }

  doubleVal = atof(argv[1]);
  intVal = atoi(argv[2]);
  cout << doubleVal << " / " << intVal << = " << (doubleVal / intVal) << endl;

  return 0;
}
```

```
cmdPrompt% cstrConvDemo
Usage: ./cstrConvDemo <floatVal> <intVal>
cmdPrompt% cstrConvDemo 92.75 16
92.75 / 16 = 5.79688
cmdPrompt%  cstrConvDemo hello 18
0 / 18 = 0
cmdPrompt%  cstrConvDemo 92.75 16.75
92.75 / 16 = 5.79688
```

Bad # params

Good run

Non-parseable values typically result in 0

atoi converts only as much as it can to an int