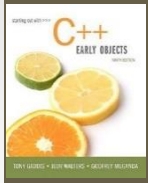


COP2334

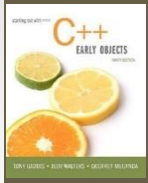
Introduction to Object Oriented Programming with C++

D. Singletary

Module 10

Ch. 12 C-Strings and the string Class

FLORIDA STATE COLLEGE at Jacksonville



- A C-string is a sequence of characters stored in consecutive memory locations and terminated by a null character
  - C++ provides the string class which provides functions and safeguards which make it preferable over C-strings

    -but-
  - Many legacy programs use C-strings
  - Low-level code uses C-strings to avoid overhead of string objects

- The type of a C-string is char* (pointer to char)

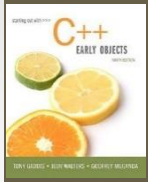
```
const char *c = "COP2334";
```

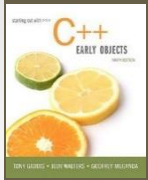
0	1	2	3	4	5	6	7
C	O	P	2	3	3	4	\0

```
// cstring.cpp
#include <iostream>
#include <cstring> // for strlen and other C-string functions
using namespace std;

int g_x = 0;

int main()
{
   const char *STR_C = "COP2334";
   const string stringVar = "COP2334";
   const char *STR_C2 = "COP2335";
   char *p = new char[10];
   char x = 'x'; // marker to help interpret memory map

   cout << "address of STR_C is " << reinterpret_cast<int>(STR_C) << endl;
   cout << "address of stringVar is " << reinterpret_cast<int>(&stringVar) << endl;
   cout << "address of STR_C2 is " << reinterpret_cast<int>(STR_C2) << endl;
   cout << "address of g_x is " << reinterpret_cast<int>(&g_x) << endl;
   cout << "address of p is " << reinterpret_cast<int>(p) << endl;
   cout << "address of x is " << reinterpret_cast<int>(&x) << endl;

   cout << "length of string using string class is " << stringVar.length() << endl;
   cout << "length of string using C-string strlen function is " << strlen(STR_C) << endl;

   cout << "C-string memory map:" << endl;
   for (int i = 0; i < strlen(STR_C); i++)
      cout << "    " << STR_C[i] << " is stored at " << reinterpret_cast<int>(STR_C + i) << endl;

   cout << "address of an anonymous literal is " << reinterpret_cast<int>(&("Hello World")) << endl;

   return 0;
}
```

```
// cstring.cpp
#include <iostream>
#include <cstring> // for strlen and other C-string functions
using namespace std;

int g_x = 0;

int main()
{
   const char *STR_C = "COP2334";
   const string stringVar = "COP2334";
   const char *STR_C2 = "COP2335";
   char *p = new char[10];
   char x = 'x'; // marker to help interpret memory map

   cout << "address of STR_C is " << reinterpret_cast<int>(STR_C) <<
   cout << "address of stringVar is " << reinterpret_cast<int>(&strin
   cout << "address of STR_C2 is " << reinterpret_cast<int>(STR_C2)
   cout << "address of g_x is " << reinterpret_cast<int>(&g_x) << en
   cout << "address of p is " << reinterpret_cast<int>(p) << endl;
   cout << "address of x is " << reinterpret_cast<int>(&x) << endl;

   cout << "length of string using string class is " << stringVar.length() << endl;
   cout << "length of string using C-string strlen function is " << strlen(STR_C) << endl;

   cout << "C-string memory map:" << endl;
   for (int i = 0; i < strlen(STR_C); i++)
      cout << "    " << STR_C[i] << " is stored at " << reinterpret_cast<int>(STR_C + i) << endl;

   cout << "address of an anonymous literal is " << reinterpret_cast<int>(&("Hello World")) << endl;

   return 0;
}
```

```
address of STR_C is 4718629
address of stringVar is 7012056
address of STR_C2 is 4718637
address of g_x is 4759560
address of p is 8459816
address of x is 7012055
length of string using string class is 7
length of string using C-string strlen function is 7
C-string memory map:
    C is stored at 4718629
    O is stored at 4718630
    P is stored at 4718631
    2 is stored at 4718632
    3 is stored at 4718633
    3 is stored at 4718634
    4 is stored at 4718635
address of an anonymous literal is 4718901
```



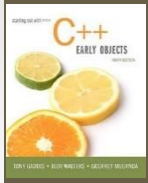


MEMORY SEGMENT	VARIABLE
STACK	x stringVar
NAMED LITERALS "ANONYMOUS" LITERALS	STR_C STR_C2 "Hello World"
GLOBALS	g_x
HEAP	p

```
address of STR_C is 4718629
address of stringVar is 7012056
address of STR_C2 is 4718637
address of g_x is 4759560
address of p is 8459816
address of x is 7012055
length of string using string class is 7
length of string using C-string strlen function is 7
C-string memory map:
   C is stored at 4718629
   O is stored at 4718630
   P is stored at 4718631
   2 is stored at 4718632
   3 is stored at 4718633
   3 is stored at 4718634
   4 is stored at 4718635
address of an anonymous literal is 4718901
```

- C-string literals are constant C-strings stored in a reserved memory segment
  - allocated implicitly by the compiler
- Variable C-strings are declared as arrays of characters -- either on the stack or from the heap
  - allocated explicitly by the programmer

```
const int SIZE = 20; // max length = 19
char company[SIZE];  // stack
char* companyH = new char[SIZE];  // heap
```



C-string Library Functions

- #include <cstring> // must include the header

- strlen() is passed a C-string and returns the length as an integer

  strlen("COP2334") returns 7

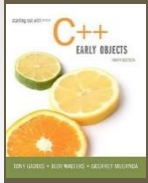
- strcat() concatenates one string onto the end of another

  char str1[13] = "Hello ";

  char str2[] = "World!";

  strcat(str1, str2) returns "Hello World!"

// str1 must be large enough to hold both, plus
// an extra character for the null terminator



- strcmp() compares two C-strings and returns 0 if the strings are equal, non-zero if not

strcmp("COP2334", "COP2335") returns -1

char s1[] = "COP2334";

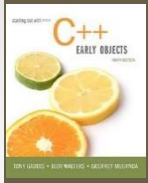
strcmp("COP2334", s1) returns 0

- The logical NOT operator can be used instead of direct comparison to 0

if (!strcmp("COP2334", s1))

is equivalent to

if (strcmp("COP2334", s1) == 0)

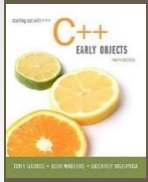


- strcpy() copies one C-string into another:

```
char  s1[] = "COP2334";
char s2[strlen(s1) + 1];  // + 1: space for \0
strcpy(s2, s1);  // copies s1 into s2
```

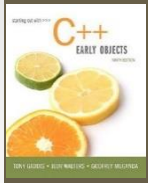
strcpy(to_string, from_string)

- to_string must be large enough to hold from_string plus the null terminator



Converting a C-string to a std::string

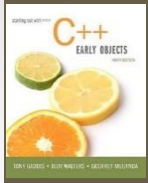
```
// convert_cstring.cpp
// demo of conversion of C-strings to std::string objects

#include <iostream>
#include <cstring>
#include <string>

using namespace std;

int main()
{
      int SIZE = 20;
      char *s1 = new char[SIZE];

      strcpy(s1, "This is a C-string");  // length = 18
      string str_s1 = s1; // '=' operator overload

      cout << "Here is the C-string: " << s1 << endl;
      cout << "Here is the string: " << str_s1 << endl;

      return 0;
}
```



Operator Overloading

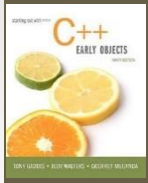
string str_s1 = s1; // '=' operator overload

- Assigning a C-string to a string object uses a technique known as an operator overload
- Existing memory is deleted and new memory is allocated for the string, then the contents of the source are copied
- We will look at overloading operators in an upcoming course unit.



String Streams: ostringstream

- ostringstream is a subclass of ostream (cout's parent class)
  - uses stream insertion operator (<<) to convert numeric values to string
  - works the same way as cout, but data is written to a string object
  - numeric-to-string conversion are performed as necessary
  - #include <sstream> header

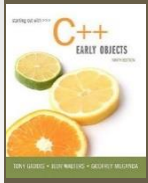


String Streams: istringstream

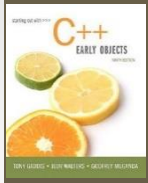
- istringstream derives from istream
  - uses stream extraction operator (>>) to read from an associated string object

```
string str = "John    20 50";
istringstream istr1(str);  // will read from str
istringstream istr2;

const char *cstr = "Amy 30 42";
istr2.str(cstr);  // will read from cstr
```

  - string-to-numeric conversions are performed as necessary
  - #include <sstream> header (same as for istringstream)

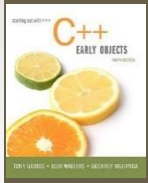


```
// This program illustrates the use of sstream objects
#include <sstream>
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string str = "John  20 50";  // String to read from
   const char *cstr = "Amy 30 42";    // Cstring to read from
   istringstream istr1(str);    // istr1 will read from str
   istringstream istr2;        // istr2 will read from cstr
   ostringstream ostr;         // The ostringstream object to write to

   string name;
   int score1, score2, average_score;

   // Read name and scores and compute average then write to ostr
   istr1 >> name >>  score1 >> score2;
   average_score = (score1 + score2)/2;
   ostr <<  name << " has average score " << average_score << "\n";
```



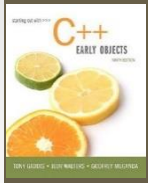
```
    // Set istr2 to read from the C string and repeat the above
    istr2.str(cstr);
    istr2 >> name >>  score1 >> score2;
    average_score = (score1 + score2)/2;
    ostr <<  name << " has average score " << average_score << "\n";

    // Switch to hexadeximal output on ostr
    ostr << hex;

    // Write Amy's scores in hexadecimal
    ostr << name << "'s scores in hexadecimal are: " << score1
        << " and " << score2 << "\n";

    // Extract the string from ostr and print it to the screen
    cout << ostr.str();

    return 0;
}
```

```
John has average score 35
Amy has average score 36
Amy's scores in hexadecimal are: 1e and 2a
```



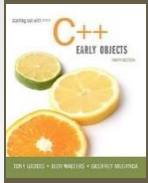
string Constructors

Definition	Description
string()	Default constructor; creates an empty string
string(const char *s)	Convert constructor; creates a string object from a C-string
string(const string &s)	Copy constructor; creates a new string from an existing string



Overloaded string Operators

OPERATOR	MEANING
>>	reads a whitespace-delimited string into a string object
<<	inserts a string object into a stream
=	assigns string on right to string object on left
+=	appends string on the right to the end of contents of string on left

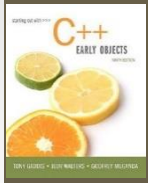


Overloaded string Operators (cont)

OPERATOR	MEANING
+	Returns concatenation of the two strings
[]	references character in string using array notation
>, >=, <, <=, ==, !=	relational operators for string comparison. Return true or false

string Member Functions

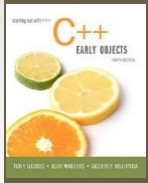
CATEGORY	FUNCTIONS
Conversion to C-String	data
Modification	append, assign, clear, copy, erase, insert, replace, swap
Space management	capacity, empty, length, size
Substrings	find, substr
Comparison	compare



Converting to C-Strings

- data() function returns the C-string equivalent of a string object
  - Useful when using a string object with a function that is expecting a C-string

```
char greeting[20] = "Have a ";
string str("nice day");
strcat(greeting, str.data());

// greeting now contains "Have a nice day"
// must be large enough to contain
// all characters + 1 for null byte
```

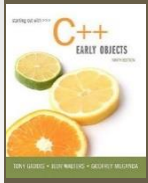


Modifying string Objects

str.append(string s)

- appends contents of s to end of str
- conversion constructor allows a C-string to be passed in place of s

```
string str("Have a ");
str.append("nice day");
// str now contains "Have a nice day"
```



Modifying string Objects (cont)

str.insert(int pos, string s)

- inserts s at position pos in str
- conversion constructor allows a C-string to be passed in place of s

```
const char* c = "very ";
string str("Have a day");
str.insert(7, "nice ");
cout << str << endl;
str.insert(7, c);
cout << str << endl;
```

Have a nice day
Have a very nice day