String & Console IO

Computer Programming for Engineers (DSAF003-42)

Instructor:

Youngjoong Ko (nlp.skku.edu)

Recap

- **Call-by-reference**
  - Pointers or references
- **const reference parameters**
- **Function Overloading**
  - Same function name, different parameters
  - Exact match first, compatible match next
- **Default Arguments**
  - Only on function definitions

STRING IN C++

Introduction

- **Review on C-strings**
  - Array with base type `char`
  - End of string marked with NULL or "\0"
  - "Older" method inherited from C
- **New String class (C++)**
  - Uses templates

STRING BASICS

C-Strings

- **Array with base type *char***
  - One character per indexed variable
  - One extra character: "\0"
    - Called "null character"
    - End-of-string marker
    - Crucial to find the length of a string
- **We have used C-strings**
  - Literal "Hello" stored as c-string

C-String Variable

- **Array of characters:**

```
char s[10] = "Hi Mom!";
```

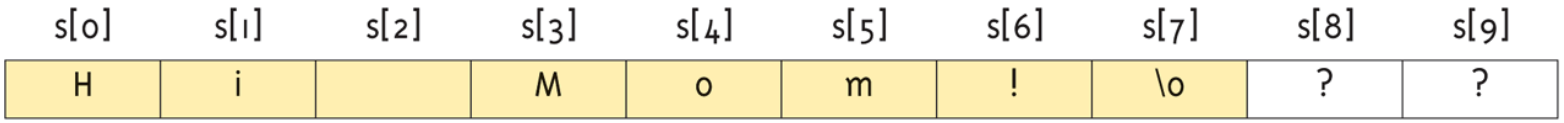
  - Declares a c-string variable to hold up to 9 characters
  - + one null character " \0"
  - Initialization places "\0" at end

- **Typically "partially-filled" array**
  - Declare large enough to hold max-size string
  - Prone to overflow error
  - Must contain null character

C-String Storage

- **A standard array**
  - If s contains string "Hi Mom!", stored as:

```
char s[10] = "Hi Mom!";
```



C-String Initialization

- **Can omit array-size:**

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

  - Automatically makes size one more than length of quoted string

Library

- **Declaring C-strings**
  - Requires no C++ library
  - Built into standard C++

- **Manipulations**
  - Require library <string.h> or <cstring> (std::* versions)
  - strcpy, strlen, str*, ... functions are included.

char overflow (1_char_overflow.cpp)


```
#include <iostream>

using namespace std;

int main()
{
  // Following code has errors
  char name1[5] = “Tazan”;
  char name2[] = “Tszan”;

  cout << name1 << “ “ << sizeof(name1) << endl;
  cout << name2 << “ “ << sizeof(name2) << endl;
}
```

char (2_char.cpp)


```
#include <iostream>

using namespace std;

int main()
{
  char s[10] = "Hi Mom!";

  cout << s << endl;

  // Checking items of s
  for(char c : s)
    cout << c << ":" << int(c) << endl;
}
```

char initialization (3_char_init.cpp)


```
#include <iostream>

using namespace std;

int main()
{
  char s1[10];
  char s2[10] = "";
  char s3[10] = {};
  char s4[10] = {'\0'};

  // Checking items of s
  cout << "s1" << endl;
  for(char c : s1)
    cout << int(c) << " ";
  cout << endl;

  cout << "s2" << endl;
  for(char c : s2)
    cout << int(c) << " ";
  cout << endl;

  cout << "s3" << endl;
  for(char c : s3)
    cout << int(c) << " ";
  cout << endl;

  cout << "s4" << endl;
  for(char c : s4)
    cout << int(c) << " ";
  cout << endl;

}
```

short string (4_short_string.cpp)


```
#include <iostream>

using namespace std;

int main()
{
  char short_string1[] = "abcdefg";
  cout << "String1: " << short_string1 << endl;
  cout << "Size: " << sizeof(short_string1) << endl;

  // What happens with the below code?
  char short_string2[] = {'k', 'l', 'm'};
  // error with ""
  //char short_string2[] = {"k", "l", "m"};
  cout << "String2: " << short_string2 << endl;
  cout << "Size: " << sizeof(short_string2) << endl;
}
```

C++ STRING CLASS

Standard Class string

- **C++ has a data type of "string" to store sequences of characters**
  - Not a primitive data type
  - Must add #include <string> at the top of the program

```
#include <string>

std::string s; // with std namespace
```

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

string s; // without std namespace
```

- **Many operators and functions defined for manipulation**
  - Same operations available as C-strings, and more!
  - Over 100 members of standard string class

Standard Class string

- **Example member functions:**
  - can assign, compare, add:
  - `.length()`: returns length of string variable
  - `.at(i)`: returns reference to char at position `i`

```
string s1, s2, s3;
s3 = s1 + s2;       // concatenation (as strcat)
s3 = "Hello Mom!"; // assignment (as strcpy)
```

  - the “+” operator on strings concatenates two strings together

Member Functions

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

Display 9.7 Member Functions of the Standard Class `string`

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

(continued)

Member Functions

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

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

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

str.length(): Returns the length of str

Standard Class string

- **Display 9.4: Program Using the Class string**

```
// demonstrates the standard class string.
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string phrase; // initialized to the empty string

    // two equivalent ways of initializing a string variable
    string adjective("fried"), noun("ants");
    string wish = "Bon appetite!";

    phrase = "I love " + adjective + " " + noun + "!";
    cout << phrase << endl << wish << endl;
    return 0;
}
```

It may work without including, BUT it is not guaranteed.

```
I love fried ants!
Bon appetite!
```

C-string and string Object Conversions

- **From C-string to string object**
  - Automatic type conversions

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

- **From string object to C-string**
  - string object cannot be assigned, since it's not a pointer must use explicit conversion:

```
strcpy( aCString, stringVar.c_str() );
aCString = stringVar; // illegal!
```

Conversion between string and numbers

- **String to 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");  // string "35" to an integer 35
d = stod("2.5"); // string "2.5" to the double 2.5
```

- **Numbers to string**

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

obj conversion (5_obj_conversion.cpp)


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

using namespace std;

int main() {
  char aCString[] = "My C-string";
#if 1
  string stringVar = aCString;

  cout << "C String: " << aCString << endl;
  cout << "string: " << stringVar << endl;

#else
  string stringVar = "C++";
  strcpy( aCString, stringVar.c_str() );
  //aCString = stringVar; // illegal!
  //aCString = stringVar.c_str(); // What happens with this code?

  cout << "C String: " << aCString << endl;
  cout << "string: " << stringVar << endl;
#endif
}
```

string example (6_string_example.cpp)


```
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string str("abcde");

  str.erase(0,1);
  cout << str << endl;
}
```

CONSOLE I/O

Console I/O

- **I/O objects cin and cout**
  - Defined in the C++ library called <iostream>
  - Must have these lines (called preprocessor directives) near start of file:

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

  - Tells C++ to use library so we can use the I/O objects cin and cout

Input Using cin

- **cin for input, cout for output**
- **Differences:**
  - ">>" (extraction operator) points opposite
    - Think of it as "pointing toward where the data goes"
  - Object name "cin" used instead of "cout"
  - No literals allowed for cin
    - Must input "to a variable"
- **cin >> num;**
  - Waits on-screen for keyboard entry
  - Value entered at keyboard is "assigned" to num

Prompting for Input: cin and cout

- **Always "prompt" user for input**

```
cout << "Enter number of dragons: ";
cin >> numOfDragons;
```

  - Note no "\n" in cout. Prompt "waits" on same line for input as follows:
    - `Enter number of dragons: ____`
    - Underscore above denotes where keyboard entry is made

- **In general, every cin should have cout prompt**
  - Maximizes user-friendly input/output

Console 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"

Input/Output Example

- **Using cin and cout with a string**

```
//Program to demonstrate cin and cout with strings
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string dogName;
    int actualAge;
    int humanAge;
    cout << "How many years old is your dog? ";
    cin >> actualAge;
    humanAge = actualAge * 7;
    cout << "What is your dog's name? ";
    cin >> dogName;
    cout << dogName << "'s age is approximately " <<
     "equivalent to a " << humanAge << " year old human."
     << endl;
    return 0;
}
```

Input/Output Example 2

- **Using cin and cout with a string**

```
How many years old is your dog? 5
What is your dog's name? Rex
Rex's age is approximately equivalent to a 35 year old human.
```

```
How many years old is your dog? 10
What is your dog's name? Mr. Bojangles
Mr.'s age is approximately equivalent to a 70 year old human.
```

  - "Bojangles" is not read into dogName because cin stops input at the space.

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()

cin example (7_cin_example.cpp)


```
#include <iostream>

using namespace std;

int main()
{
  int num = 0;
  // See the initial value of num
  cout << "Initial value of num: " << num << endl;
  cout << "Enter a new number for num: ";
  cin >> num;
  // What happens when we enter a value
  // with a different data type rather than int?
  cout << "Entered value of num: " << num << endl;
  return 0;
}
```

io example with getline (8_getline.cpp)


```
//Program to demonstrate cin and cout with strings
#include <iostream>
#include <string>
using namespace std;

int main( )
{
  string dogName;
  int actualAge;
  int humanAge;
  cout << "How many years old is your dog? ";
  cin >> actualAge;
  humanAge = actualAge * 7;
  cout << "What is your dog's name? ";
  // Is it enough?
  getline(cin, dogName);
  cout << dogName << "'s age is approximately " <<
    "equivalent to a " << humanAge << " year old human."
    << endl;
  return 0;
}
```

Console Output

- **What can be outputted?**
  - Any data can be outputted to display screen
    - Variables
    - Constants
    - Literals
    - Expressions (which can include all of above)

```
cout << numberOfGames << " games played.";
```

- **Cascading: multiple values in one cout**

Separating Lines of Output

- **New lines in output**
  - Recall: "\n" is escape sequence for the char "newline"

```
cout << "Hello World\n";
```

- **A second method: object std::endl**

```
cout << "Hello World" << endl;
```

  - Sends string "Hello World" to display and "\n", skipping to next line
  - Same result as above

Formatting Output

- **Formatting numeric values for output**
  - Values may not display as you'd expect!

```
cout << "The price is $" << price << endl;
```

    - If price (declared double) has value 78.5, you might get:
    - The price is $78.500000 or:
    - The price is $78.5
  - We must explicitly tell C++ how to output numbers in our programs!

Formatting Numbers

- **"Magic Formula" to force decimal sizes:**

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- **These statements force all future cout'ed values:**
  - To have exactly two digits after the decimal place

```
cout << "The price is $" << price << endl;
```

  - Now results in the following: The price is $78.50

formatting output (9_format_output.cpp)


```
#include <iostream>

using namespace std;

int main()
{
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(3);

  double price = 78.5909309283;
  cout << "The price is $" << price << endl;

  return 0;
}
```