Strings in C++

Learning Objectives

- An Array Type for Strings
- Standard Class string
  - String processing

Introduction

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

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

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;

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
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)

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
`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	
`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`.

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;
  ```

  ~~`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"
```