Ch 10. Characters, C-Strings, and the string class

CS 2308
Fall 2013
Jill Seaman

Characters

- Built-in data type
- Value: a single character
- Literals: ‘a’, ‘!’, ‘\n’, ‘8’, ...
- Operations:
  - assignment: =
  - compare: ==, <, etc.
  - conversion to/from int: uses the ascii code

```cpp
char ch = 'A';
cout << ch + 10 << endl;
cout << static_cast<char>(ch+10) << endl;
```

Output:

75

10.1 Character Testing

- The C++ library provides several functions for testing characters.
- Requires the cctype header file
- They take a char (or int) argument
- They return non-zero for true, 0 for false, and can be used as boolean expressions:

```
char input;
...
if (isupper(input)) ...
```

Character Testing

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalpha</td>
<td>true for any letter of the alphabet</td>
</tr>
<tr>
<td>isalnum</td>
<td>true for letter or digit</td>
</tr>
<tr>
<td>isdigit</td>
<td>true for digit</td>
</tr>
<tr>
<td>islower</td>
<td>true for lowercase letter</td>
</tr>
<tr>
<td>ispunct</td>
<td>true for anything not a digit, letter or space</td>
</tr>
<tr>
<td>isupper</td>
<td>true for uppercase letter</td>
</tr>
<tr>
<td>isspace</td>
<td>true for space, tab, newline (aka whitespace)</td>
</tr>
</tbody>
</table>
10.2 Character Case conversion

- These take a char (or int), and return an int(!)
- `toupper(c)`
  - if c is lowercase, returns its uppercase version
  - otherwise returns c
- `tolower(c)`
  - if c is uppercase, returns its lowercase version
  - otherwise returns c
- Does NOT change argument

```cpp
char x = 'A';
char y = tolower(x);
cout << x << " " << y << endl;
```

Output:
```
A a
```

10.3 C-Strings

- In any programming language, a “string” is a sequence of characters.
- In C++, a C-String is a certain way of representing a string in memory
- A C-String is:
  - a sequence of characters
  - stored in consecutive memory locations
  - ALWAYS terminated by a null character (\0, ascii=0)

```
Hi There!
```

C-String

- String literals are stored in memory as C-Strings:
  - “Jim Kase”, “A00123456”, “$2.35/lb”
  - they have type char[]
- A C-String can be stored in a char array.
  - Make sure array is large enough for the null char! (add one to the length).
- Do NOT pass the array size to functions taking C-strings as arguments
  - Unnecessary, because the null char marks the end.

Operations over C-Strings

- Don’t use = or == on char[] (arrays: doesn’t work)
- input: can use `>>`
  - input stops at whitespace (space, tab, newline)!
  - but copying to memory does NOT stop at end of array
- input: can use `cin.getline(char s[], int n)`
  - input stops at ‘\n’ OR after n-1 characters have been read
- output: can use `<<`

```cpp
char cstr[10];
cout << "Enter a name: " << cstr << endl;
cin.getline(cstr,10);
cout << "You entered: " << cstr << endl;
```

```
Enter a name: Tom Fox
You entered: Tom Fox
```
10.4 Library Functions for C-Strings

- Usually require the cstring header
- Function headers look like this: `func(char *s)`
  - recall `char *s` is basically equivalent to `char s[]`
- the argument can be:
  - the name of a char array
  - a literal string

C-string length

- `int strlen (char* str)`
- Returns the number of characters in a C-string (up to but not including the null char).

```
char cstr[30] = "Economics";
cout << strlen(cstr) << endl; // prints 9
```

C-string copy

- Use strcpy to perform assignment for C-strings
  ```
  char* strcpy (char *destination, char *source);
  ```
- Copies source C-string to destination
  - destination is modified
  - destination must be long enough
  - ignore returned value (destination is returned)
- example:

```
char string1[13] = "Hello ";
char string2[7] = "World!";
// simulate: string1 = string2;
strcpy(string1, string2);
cout << string1 << endl;
```

C-string compare

- Use strcmp to perform comparison for C-strings
  ```
  int strcmp (char *str1, char *str2);
  ```
- Compares str1 and str2
  - if str1 and str2 are the same, returns 0
  - if str1 comes before str2 alphabetically, returns -1
  - if str1 comes after str2 alphabetically, returns 1
- example:

```
char string1[13] = "Hello ";
char string2[7] = "World!";
// if (string1<string2)...
if (strcmp(string1, string2) < 0)
cout << "Hello comes before World" << endl;
```
10.7 More about the C++ string class

- string is a data type provided by the C++ library.
  - Specifically it is a class (see chapter 13).
- string requires the <string> header file
  - <iostream> may work as well
- To define a string variable:
  - string name1;
  - name1 is a string object.
  - initialized to the empty string (size is 0)
- The representation in memory of a string object is hidden from the programmer.

Empty string literal:
"

Operations over string objects

- initialization using = with a C-string literal or string

```
string name1 = "Steve Jobs";
string name2 = name1;
```

- assignment using = with C-string literal or string

```
string name1, name2;
name1 = "Andre Johnson";
name2 = name1;
```

Operations over string objects

- output using <<

```
string name1 = "Steve Jobs";
cout << "Name " << name1 << endl;
```

- input using >>

```
string name1;
cout << “Enter your name “;
cin >> name1;
```

- input using getline

```
string name1;
cout << “Enter your name “;
getline (cin, name1);
```

Operations over string objects

- comparing string objects: < <= > == !=
  (alphabetical order)

```
string string1, string2;
string1 = "Hello “;
string2 = "World!";
if (string1 < string2)
cout << “Hello comes before World” << endl;
```

- string objects can be compared to C-strings

```
string string1;
cout << “Enter a word: “;
cin >> string1;
if (string1 == “Hello")
cout << “You entered Hello.” << endl;
```
More operations over string objects

- \([n]\) subscript notation, returns char at position \(n\)
- or use `string.at(n)` -- performs bounds check

```cpp
string string1 = "Hello ";
cout << string1[4] << endl;
cout << string1.at(1) << endl;
```

```cpp
string1[0] = 'h'; //this works
string1[6] = 's'; //this gets ignored (6>=length)
string1.at(6) = 's'; //this causes a run-time error
```

```
string string1 = "Hello ";
cout << string1[4] << endl;
cout << string1.at(1) << endl;
```

```
string theString = "Hello ";
cout << theString.length() << endl; //outputs 5
```

```
string theString = "Hello ";
theString.append(" World");
cout << theString << endl; //outputs: Hello World
```

---

Exercise

- Write a function `countDigits` that takes a string as an argument and outputs the number of digits it contains.

```cpp
int countDigits (string p) {
    int count = 0;
    for (int i=0; i < p.length(); i++) {
        if (isdigit(p.at(i)))
            count++;
    }
    return count;
}
```

Exercise (watchout)

- Write a function `toLowerCase` that takes a string \(p\) as an argument and returns a NEW string that is a copy of \(p\) with all of its uppercase letters converted to lowercase.

```cpp
string toLowerCase (string p) {
    string newP;
    for (int i=0; i < p.length(); i++) {
        newP.at(i) = tolower(p.at(i));
    }
    return newP;
}
```

```cpp
int countDigits (string p) {
    int count = 0;
    for (int i=0; i < p.length(); i++) {
        if (isdigit(p.at(i)))
            count++;
    }
    return count;
}
```

```
string toLowerCase (string p) {
    string newP;
    for (int i=0; i < p.length(); i++) {
        newP.at(i) = tolower(p.at(i));
    }
    return newP;
}
```

```
string toLowerCase (string p) {
    string newP;
    for (int i=0; i < p.length(); i++) {
        newP.at(i) = tolower(p.at(i));
    }
    return newP;
}
```

```
string theString = "Hello ";
cout << theString.length() << endl; //outputs 5
```

```
string theString = "Hello ";
theString.append(" World");
cout << theString << endl; //outputs: Hello World
```

```
string theString = "Hello ";
cout << theString.length() << endl; //outputs 5
```

```
string theString = "Hello ";
theString.append(" World");
cout << theString << endl; //outputs: Hello World
```

```
string theString = "Hello ";
cout << theString.length() << endl; //outputs 5
```

```
string theString = "Hello ";
theString.append(" World");
cout << theString << endl; //outputs: Hello World
```

```
string theString = "Hello ";
cout << theString.length() << endl; //outputs 5
```

```
string theString = "Hello ";
theString.append(" World");
cout << theString << endl; //outputs: Hello World
```

```
string theString = "Hello ";
cout << theString.length() << endl; //outputs 5
```