

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

CS 2308
Spring 2014

Jill Seaman

1

Characters

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

```
char ch;  
ch = 'a';  
if (ch=='A') ...
```

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

Output:

```
75  
K
```

2

10.1 Character Testing

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

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

3

Character Testing

<code>isalpha</code>	true for any letter of the alphabet
<code>isalnum</code>	true for letter or digit
<code>isdigit</code>	true for digit
<code>islower</code>	true for lowercase letter
<code>ispunct</code>	true for anything not a digit, letter or space
<code>isupper</code>	true for uppercase letter
<code>isspace</code>	true for space, tab, newline (aka whitespace)

4

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

```
char x = 'A';  
char y = tolower(x); //converts to char  
cout << x << " " << y << endl;
```

Output:

```
A a
```

5

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 (`char`)
 - stored in consecutive memory locations
 - ALWAYS terminated by a null character (`'\0'`, `ascii=0`)

H	i		T	h	e	r	e	!	\0
---	---	--	---	---	---	---	---	---	----

6

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).

7

C-String

- No need to pass the array size to functions taking C-strings as arguments
 - Unnecessary, because the null char marks the end.
 - Use a sentinel-controlled loop:

```
int cStringLength (char cstr[]) {  
    int i=0;  
    while (cstr[i]<>'\0')  
        i++;  
    return i;  
}
```

8

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 <<

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

```
Enter a name:
Tom Fox
You entered: Tom Fox

Enter a name:
Tom Johnson
You entered: Tom Jøhns
```

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
```

11

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 (must be '\0' terminated!)
 - a literal string

10

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;
```

Output:

```
World!
```

12

C-string compare

- Use strcmp to perform comparison for C-strings
- `int strcmp (char *str1, char *str2);`
- Compares str1 and str2, using ascii values
 - 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;
```

13

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:

```
""
```

14

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;
```

15

Operations over string objects

- output using <<

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

- input using >> (input stops at first whitespace!)

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

- input using getline **note: not the same one as for c-strings**

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

stops at first '\n'

16

Operations over string objects

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

```
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;
```

17

More operations over string objects

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

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

Output:

o
e

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

terminate called throwing an exceptionAbort trap: 6

18

string class member functions

- string class has many member functions that operate over the string object (Table 10-7).
- `theString.length()` : returns length of string stored in theString (can also use `.size()`).

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

- `theString.append(str)`: appends str (string object or c-string) to the end of theString
 - It changes theString!! (also changes its length)

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

19

Exercise

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

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

- Now write it for C-strings.

20

Exercise (watchout)

- Write a function `toLowerCaseString` 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.

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

What is wrong with this solution?

Does it change the argument?
Is the return value deallocated?

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

What is wrong with this solution?

terminate called throwing
an exceptionAbort trap: 6