CS31 INTRODUCTION TO COMPUTER SCIENCE

FALL 2015

DISCUSSION SESSION 6

NOVEMBER 6, 2015

Noor Abani

TWO-DIMENSIONAL ARRAY

- Declared and initialized like one dimensional arrays but use nested braces for initializer list.
  - Ex: int n[][3] = {{2,1,4} ,{9,7,3}};
  - Note that you can skip the size of the first dimension (but not the second)
- Accessing elements: <name>[row_index][col_index]
  - Ex: n[0][0] is 2
  - n[1][2] is 3
- We use two nested for loops to access each element of a 2D array.

C++ STRINGS

- #include <string>
- Strings are an array of characters.
- We can access each character of a string just like we use array indices to access elements at particular positions.

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{ string hola = "hi!";
  string greets[] = {"howdy", "bonjour", "hallo"};
  cout << ??? << endl;
}
```

Q1: How can we print the '!' in hola?

A1: hola[2]

ARRAY OF STRINGS

- Thinking about strings as arrays of characters means that an array of strings is an array of an array of characters
- As such, we can access the j-th character of an i-th string in an array via the syntax:
  - stringArray[i][j]

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
     string greets[] = {"howdy", "bonjour", "hallo"};
     greets[1][0] = 'B';
     cout << greets[1] << endl;
     cout << greets[2][4] << endl;
}
```

Q: What is the output of this program?

A: Bonjour
o

C++ STRING FUNCTIONS

- length() : returns the number of characters in a string
  - Ex: string s = “cs31”;
  - s.length() will return 4
- substr(int i, int l): return the substring of s that starts at index i and has length
  - Ex: s.substr(0,1) returns “c”
  - Ex: s.substr(1, 2) return “s3”

- We can use the ‘+’ operator to concatenate two strings
- We can use the ‘==“ operator to check if two strings are equal

PRACTICE

- Write a function that takes in a string by reference and removes all the exclamation marks from it
  - void removeExclamations(string &s)
  - Ex: if s = “w!!!haa!!t!”, s becomes “whaat”

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

void removeExclamations(string &s)
{
    for (int i = 0; i < s.length(); i++)
    {
        if (s[i] == '!')
        {
            s = s.substr(0, i) + s.substr(i + 1, s.length() - i - 1);
            i--;
        }
    }
}
```

REMEMBER- ASCII CHARACTER SET

32		56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	"	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	'	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	-	69	E	93	]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	`	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	|
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g		

CHARACTER OPERATIONS

- We can compare two characters … What we are actually comparing is the ASCII code of the characters.
- Notice that the ascii code of lower case letter is greater than that of upper case ones;
  - ‘a’ > ‘A’
  - ‘a’ > ‘Z’
  - ‘O’ < f

USEFUL FUNCTIONS ON CHARACTERS

- #include <cctype>
- **Character Classification:**
  - **isalpha** asks if this character is a letter.
  - **isdigit** asks if this character is a number.
  - **isalnum** asks if this character is alphanumeric. (is it a number or a letter?)
  - **isupper** asks if this character is an uppercase letter.
  - **islower** asks if this character is a lowercase letter.

EXAMPLES

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;
int main ()
{
     char numInChar = '5';
     cout << isalnum(numInChar) <<
endl;
}
```

Output: some non-zero number

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;
int main ()
{
     char numInChar = '5';
     cout << isupper(numInChar) <<
endl;
}
```

Output: 0

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;
int main ()
{
     string stringyNum = 'hi5';
     cout << isalnum(stringyNum[1]) <<
          endl;
}
```

The program will not compile. We should use double quotes for a string.

MORE FUNCTIONS

- **Character Conversions:**
  - **tolower** converts the given character to a lowercase version of itself.
  - **toupper** converts the given character to an uppercase version of itself.
- Write a program that queries the user for a string, swaps all lowercase letters for uppercase ones (and vice-versa), and then prints out the result.
  - void flip(string &s)
  - Ex: if s = “What’s Up!”, s becomes “wHAT’S uP!”

SOLUTION

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

void flip(string &s)
{
    for (int i = 0; i< s.length(); i++)
    {
        if (islower(s[i]))
        {
            s[i] = toupper(s[i]);
        }
        else if (isupper(s[i]))
        {
            s[i] = tolower(s[i]);
        }
    }
}
```

CSTRINGS

- Many libraries (especially legacy ones) that you may wish to interface with will use cstrings and you need to know how they're different from C++ strings.
- What are the main differences between C++ strings and cstrings?
  - cstrings have a null-terminating character indicating their conclusion, C++ strings do not.
  - cstrings require the programmer to define their space allocation, C++ strings allocate memory dynamically.
  - cstring functions are from the cstring library, but string functions are from the string library.
- A string that is "null-terminated" means that the end of the string is indicated by the 0-byte, AKA the null character, null terminator, etc.
- The null character, regardless of character encoding schema, is always represented as '\0' and has character code 0.

CSTRINGS

- **Declaring cstrings:**
  We can declare cstrings just as we would any other type of array:
  char <name>[size];
- **Initializing cstrings:**
  We can initialize cstrings using a variety of tactics, but the best way is:
  char <name>[] = "<string_to_initialize>";
- Remember that because cstrings are just arrays of characters, we MUST ust define size at compile time.
- Because cstrings are null-terminated, we must leave space for the extra 0-byte at the end.

EXAMPLES

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char c[3];
    cout << c << endl;
}
```

Output: some junk values

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char c[3] = "";
    cout << c<< endl;
}
```

Output: the empty string

EXAMPLES

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char c[3] = "cat";
    cout << c << endl;
}
```

This program will not compile. The space allocated is not enough because it does not allocate space for the null terminating character.

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char c[3] = "ca\0";
    cout << c<< endl;
}
```

This program will not compile. The space allocated is not enough because it does not allocate space for the null terminating character.

EXAMPLES

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char c[6] = "c\0a\0t";
    cout << c<< endl;
    cout << c[2] << endl;
}
```

Output: c
a

MORE ON INITIALIZATION

- we could also initiate cstrings using the array curly-bracket notation.

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char inTheWorld[] = {'y', 'e', 's', '!'};
    cout << inTheWorld << endl;
}
```

Output: yes! with some garbage because we did not add the null terminating character.

fix

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char inTheWorld[] =
{'y', 'e', 's', '!' ,'\0'};
    cout << inTheWorld <<
endl;
}
```

CSTRING LIBRARY

- All of those nice tricks with strings being able to compare equivalence using ==, get the length using .length(), etc.?
- We need to learn them with different names and constraints for cstrings
- Strlen for the length
- Strcmp to compare two strings
- Strcpy to copy one cstring to another

STRLEN

- **strlen** or "string length" behaves similarly the stl string .length() method. It returns the number of characters before the first null character in a cstring.

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    char strlenMeBro[] = "sup";
    char strlenSux[] = "hey\0you";
    char imSoEmpty[] = {'\0'};
    cout << strlen(strlenMeBro) << endl;
    cout << strlen(strlenSux) << endl;
    cout << strlen(imSoEmpty) << endl;
}
```

Output: 3
3
0

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    string strlenMeBro= "sup";
    cout << strlen(strlenMeBro) <<
endl;
}
```

STRCMP

- **strcmp** or "string compare" behaves differently than the stl string comparison operators like ==, <, >=. It returns a number that determines if the first argument is less than (some value < 0), greater than (some value > 0), or equal to (= 0) the second.
- We start at each cstring's first element and compare the two element by element.
  - If the two elements are the same character, keep looking down each cstring. If they both null-terminate at the same time, return 0.
  - If the two elements differ, and the first one is a lesser character code than the second, return some int less than 0.
  - If the two elements differ, and the first one is a greater character code than the second, return some int greater than 0

STRCMP EXAMPLES

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    char strcmpMeBro[]= "sup";
    char strcmpMeToo[] = "sup";
    cout << strcmp(strcmpMeBro, strcmpMeToo) << endl;
}
```

Both program output 0

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
        char strcmpMeBro[]= "sup";
        cout << strcmp(strcmpMeBro, "sup") <<
    endl;
}
```

STRCMP EXAMPLES

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    char strcmpMeBro[]= "sup";
    char strcmpSux[] = "sup\0man";
    cout << strcmp(strcmpMeBro, strcmpSux) << endl;
    cout << strcmp(strcmpSux, strcmpMeBro) << endl;
}
```

Output: 0
0

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    char strcmpMeBro[]= "sup";
    char strcmpLOUD[] = "SUP";
    cout << strcmp(strcmpMeBro, strcmpLOUD) << endl;
    cout << strcmp(strcmpLOUD, strcmpMeBro) << endl;
}
```

Output: 1
-1

STRCPY

- **strcpy** or "string copy" takes the cstring in the source and copies it into the destination.
- The function parameters request the destination **first**, followed by the source.

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    char strcpyMeBro[]= "sup";
    char copyToHere[4];
    strcpy(copyToHere, strcpyMeBro);
    strcpyMeBro[0] = 'S';
    strcpyMeBro[1] = 'U';
    strcpyMeBro[2] = 'P';
    cout << copyToHere << endl;
    cout << strcpyMeBro << endl;
}
```

Output: sup
SUP

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    char strcpyMeBro[]= "sup";
    char copyToHere[4] = strcpy
    strcpyMeBro[0] = 'S';
    strcpyMeBro[1] = 'U';
    strcpyMeBro[2] = 'P';
    cout << copyToHere << endl;
    cout << strcpyMeBro << endl;
}
```

This program will not compile.
Assignment is not allowed!

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main ()
{
    char dessert[]= "banana!";
    char stomach[7];
    strcpy(stomach, dessert);
    cout << stomach << endl;
 }
```

Error! We did not reserve enough space!

PRACTICE

- Design a function, csReverse, that takes in a cstring and reverses the order of characters in it
  - void csReverse(char c[])

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

void csReverse(char c[])
{
    int len = strlen(c);
    for (int i = 0; i < (len / 2); i++)
    {
        char temp = c[i];
        c[i] = c[len - 1 - i];
        c[len - 1 - i] = temp;
    }
}
```