CSC180: Lecture 22

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

True and False in C

- there is no boolean datatype…
- In C: The number 0 is considered to be false and all other numbers are considered to be true
- Examples:
  - if(1 == 1) //true
    if(1 != 1) //false
    if(i = 1) //true
    if(i = 0) //false
    if(i = 1 + 1) //true

- <string.h> is full of string manipulation functions

FUNCTION	DESCRIPTION	CAUTIONS
`strcpy(`*Target_String_Var*`,` *Src_String*`)`	Copies the C-string value *Src_String* into the C-string variable *Target_String_Var*.	Does not check to make sure *Target_String_Var* is large enough to hold the value *Src_String*.
`strcpy(`*Target_String_Var*`,` *Src_String, Limit*`)`	The same as the two-argument `strcpy` except that at most *Limit* characters are copied.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcpy`. Not implemented in all versions of C++.
`strcat(`*Target_String_Var*`,` *Src_String*`)`	Concatenates the C-string value *Src_String* onto the end of the C-string in the C-string variable *Target_String_Var*.	Does not check to see that *Target_String_Var* is large enough to hold the result of the concatenation.

(continued)

- <string.h> is full of string manipulation functions

FUNCTION	DESCRIPTION	CAUTIONS
`strcpy(`*Target_String_Var*`,` *Src_String*`)`	Copies the C-string value *Src_String* into the C-string variable *Target_String_Var*.	Does not check to make sure *Target_String_Var* is large enough to hold the value *Src_String*.
`strcpy(`*Target_String_Var*`,` *Src_String, Limit*`)`	The same as the two-argument `strcpy` except that at most *Limit* characters are copied.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcpy`. Not implemented in all versions of C++.
`strcat(`*Target_String_Var*`,` *Src_String*`)`	Concatenates the C-string value *Src_String* onto the end of the C-string in the C-string variable *Target_String_Var*.	Does not check to see that *Target_String_Var* is large enough to hold the result of the concatenation.

(continued)

FUNCTION	DESCRIPTION	CAUTIONS
`strcat(`*Target_String_Var*`,` *Src_String, Limit*`)`	The same as the two argument `strcat` except that at most *Limit* characters are appended.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcat`. Not implemented in all versions of C++.
`strlen(`*Src_String*`)`	Returns an integer equal to the length of *Src_String*. (The null character, `'\0'`, is not counted in the length.)	
`strcmp(`*String_1*`,` *String_2*`)`	Returns `0` if *String_1* and *String_2* are the same. Returns a value < 0 if *String_1* is less than *String_2*. Returns a value > 0 if *String_1* is greater than *String_2* (that is, returns a nonzero value if *String_1* and *String_2* are different). The order is lexicographic.	If *String_1* equals *String_2*, this function returns `0`, which converts to `false`. Note that this is the reverse of what you might expect it to return when the strings are equal.
`strcmp(`*String_1*`,` *String_2, Limit*`)`	The same as the two-argument `strcat` except that at most *Limit* characters are compared.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcmp`. Not implemented in all versions of C++.

= and C-strings

- C-strings not like other variables
  - Cannot assign or compare:
    char aString[10];
    aString = "Hello"; // ILLEGAL!
    - Can ONLY use "=" at declaration of c-string!

- Must use library function for assignment:
  strcpy(aString, "Hello");
  - Built-in function (in string library)
  - Sets value of aString equal to "Hello"
  - NO checks for size!
    - Up to programmer, just like other arrays!

Comparing C-strings

- cannot use operator ==

```
char aString[10] = "Hello";
char anotherString[10] = "Goodbye";
```

  - if(aString == anotherString) // NOT allowed!

- Must use library function:

```
if (strcmp(aString, anotherString))
      printf( "Strings same. " );
else
       printf("Strings are NOT same.");
```

C-string Functions: strlen()

- "String length"
- Often useful to know string length:
  char myString[10] = "dobedo";
  printf (“ %d”, strlen(myString));
  - Returns number of characters
    - Not including null
  - Result here:
    6

C-string Functions: strcat()

- strcat()
- "String concatenate":
  char stringVar[20] = "The rain";
  strcat(stringVar, "in Spain");
  - Note result:
    stringVar now contains "The rainin Spain"
  - Be careful!
  - Incorporate spaces as needed!

```
char studentName[21];
char myname[16];
char yourname[16];
```

Statement	Effect
`strcpy(myname, "John Robinson");`	`Myname = "John Robinson"`
`strlen("John Robinson");`	Returns 13, the length of the string "John Robinson"
`int len;` `len = strlen("Sunny Day");`	Stores 9 into len
`strcpy(yourname, "Lisa Miller");` `strcpy(studentName, yourname);`	`yourname = "Lisa Miller"` `studentName = "Lisa Miller"`
`strcmp("Bill", "Lisa");`	Returns a value < 0

`strcpy(yourname, "Kathy Brown");`
`strcpy(myname, "Mark G. Clark");`
`strcmp(myname, yourname);`
	`yourname = "Kathy Brown"`
	`myname = "Mark G. Clark"`
	Returns a value > 0

C-string Arguments and Parameters

- Recall: c-string is array
- So c-string parameter is array parameter
  - C-strings passed to functions can be changed by receiving function!
- Like all arrays, typical to send size as well
  - Function "could" also use "\0" to find end
  - So size not necessary if function won’t change c-string parameter
  - Use "const" modifier to protect c-string arguments

C-String input

- A string can be read from the user using %s

```
char president[20];
scanf (“%s”, president);
```