CSC180: Lecture 21

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

C Strings

C Strings

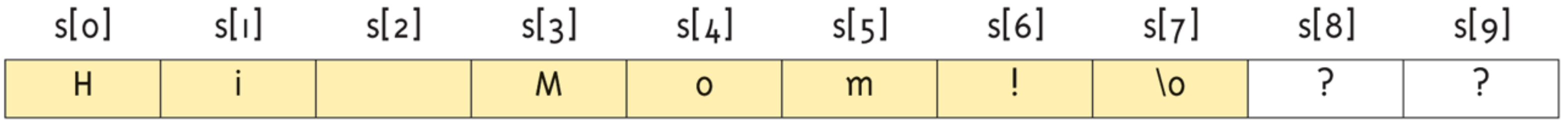
- Character array - an array whose components are of type char
- String - a sequence of zero or more characters enclosed in double quote marks “”
  - C stings are null terminated (‘\0’)
  - The last character in a string is the null character

C-String Variable

- Array of characters:
  char s[10];
  - Declares a c-string variable to hold up to 9 characters
  - + one null character
- Typically "partially-filled" array
  - Declare large enough to hold max-size string
  - Indicate end with null
- Only difference from standard array:
  - Must contain null character

C-String Storage

- A standard array:
  char s[10];
  - If s contains string "Hi Mom!", stored as:



s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	i		M	o	m	!	\0	?	?

C-strings: declaring and initializing

- Using arrays:
    - char message[] = {'H', 'i', '!', '\0'};
    - char message[] = {"Hi!"};

- Using pointers
    - const char *message = "Hi!";
    - char *message = new char[10];

C Strings (continued)

- There is a difference between `'A'` and `"A"`
    - `'A'` is the character A
    - `"A"` is the string A
- Because strings are null terminated, `"A"` represents two characters, `'A'` and `'\0‘`
- Similarly, `"Hello"` contains six characters, `'H'`, `'e'`, `'l'`, `'l'`, `'o'`, and `'\0'`

C Strings (continued)

- Consider the statement

```
      char name[16] = “hello”;
```

- Because C strings are null terminated and name has sixteen components
  - the largest string that can be stored in name is 15
- If you store a string of length, say 10 in name
  - the first 11 components of name are used and the last 5 are left unused

C Strings (Character Arrays)

- The statement

  ```
  char name[16] = "John";
  ```

  declares a string variable `name` of length 16 and stores `"John"` in it

- The statement

  ```
  char name[] = "John";
  ```

  declares a string variable name of length 5 and stores `"John"` in it

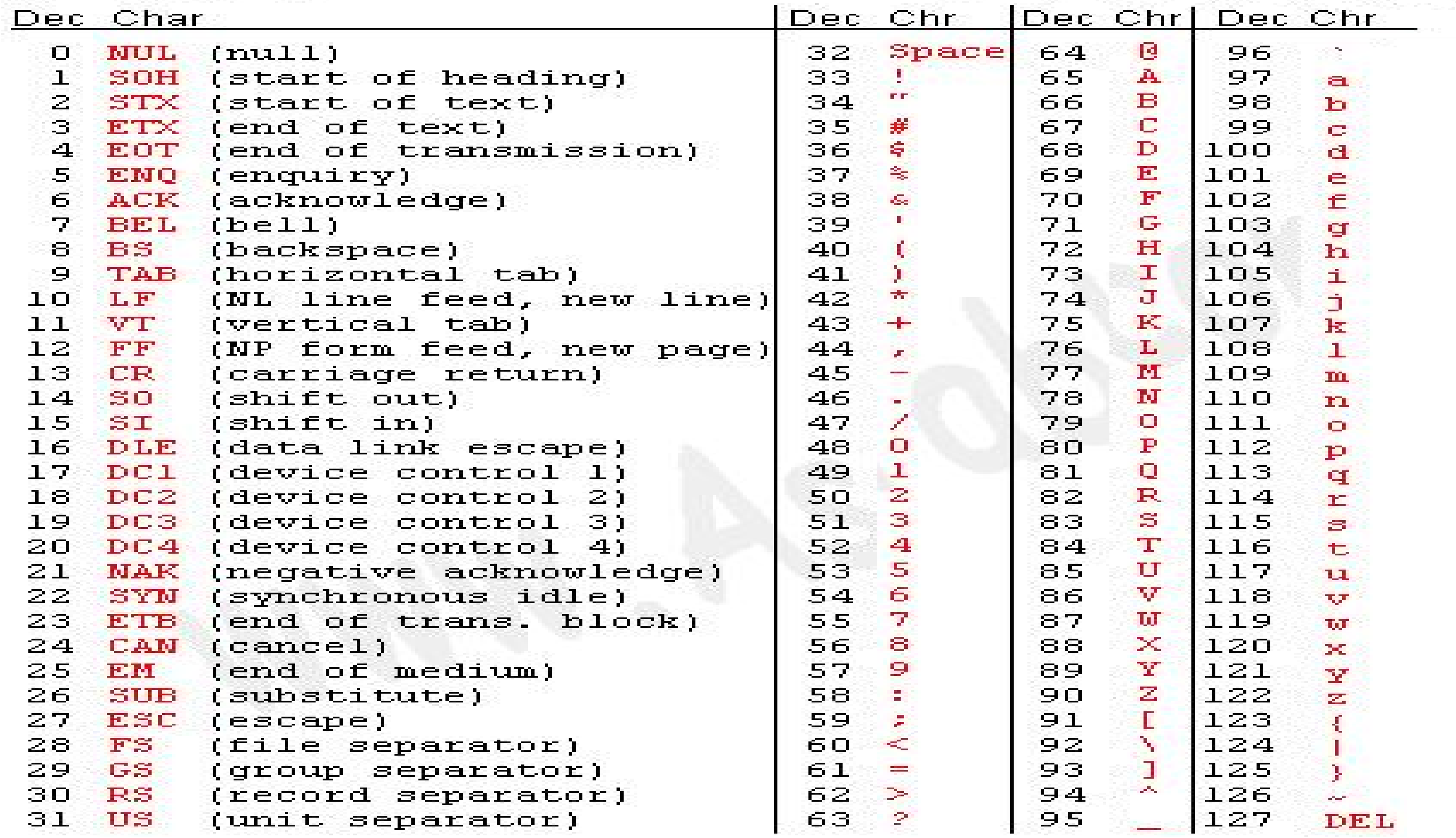
String Comparison

- C-strings are compared character by character using the collating sequence of the system
- If we are using the ASCII character set
  1. The string `"Air"` is smaller than the string `"Boat"`
  2. The string `"Air"` is smaller than the string `"An"`
  3. The string `"Bill"` is smaller than the string `"Billy"`
  4. The string `"Hello"` is smaller than `"hello"`

String Comparison: uses ASCII values

"Hello" is smaller than "hello“

"Air"   is smaller than the string "Boat"



Dec	Char		Dec	Chr	Dec	Chr	Dec	Chr
0	NUL	(null)	32	Space	64	@	96	`
1	SOH	(start of heading)	33	!	65	A	97	a
2	STX	(start of text)	34	"	66	B	98	b
3	ETX	(end of text)	35	#	67	C	99	c
4	EOT	(end of transmission)	36	$	68	D	100	d
5	ENQ	(enquiry)	37	%	69	E	101	e
6	ACK	(acknowledge)	38	&	70	F	102	f
7	BEL	(bell)	39	'	71	G	103	g
8	BS	(backspace)	40	(	72	H	104	h
9	TAB	(horizontal tab)	41	)	73	I	105	i
10	LF	(NL line feed, new line)	42	*	74	J	106	j
11	VT	(vertical tab)	43	+	75	K	107	k
12	FF	(NP form feed, new page)	44	,	76	L	108	l
13	CR	(carriage return)	45	-	77	M	109	m
14	SO	(shift out)	46	.	78	N	110	n
15	SI	(shift in)	47	/	79	O	111	o
16	DLE	(data link escape)	48	0	80	P	112	p
17	DC1	(device control 1)	49	1	81	Q	113	q
18	DC2	(device control 2)	50	2	82	R	114	r
19	DC3	(device control 3)	51	3	83	S	115	s
20	DC4	(device control 4)	52	4	84	T	116	t
21	NAK	(negative acknowledge)	53	5	85	U	117	u
22	SYN	(synchronous idle)	54	6	86	V	118	v
23	ETB	(end of trans. block)	55	7	87	W	119	w
24	CAN	(cancel)	56	8	88	X	120	x
25	EM	(end of medium)	57	9	89	Y	121	y
26	SUB	(substitute)	58	:	90	Z	122	z
27	ESC	(escape)	59	;	91	[	123	{
28	FS	(file separator)	60	<	92	\	124	|
29	GS	(group separator)	61	=	93	]	125	}
30	RS	(record separator)	62	>	94	^	126	~
31	US	(unit separator)	63	?	95	_	127	DEL

C-String Indexes

- Recall: a c-string IS an array
- Can access indexed variables of:
  char ourString[5] = "Hi";
  - ourString[0] is "H"
  - ourString[1] is "i"
  - ourString[2] is "\0"
  - ourString[3] is unknown
  - ourString[4] is unknown

C-String Initialization

- Can initialize c-string:
  char myMessage[20] = "Hi there.";
  - Needn’t fill entire array
  - Initialization places "\0" at end
- Can omit array-size:
  char shortString[] = "abc";
  - Automatically makes size one more than length of quoted string
  - NOT same as:
    char shortString[] = {"a", "b", "c"};

C-String Index Manipulation

- Can manipulate indexed variables

```
char happyString[7] = "DoBeDo";
happyString[6] = "Z";
```

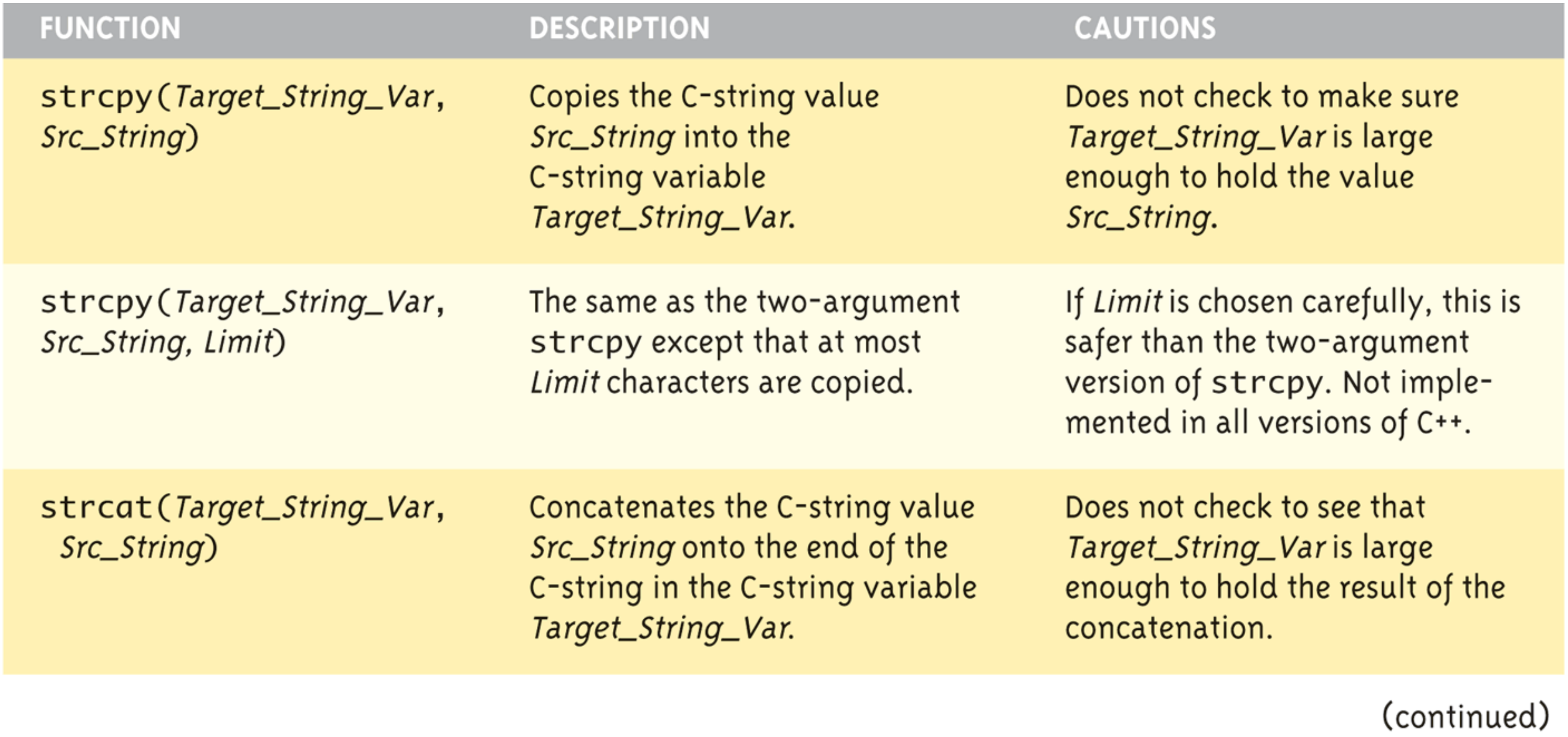
  - Be careful!
  - Here, "\0" (null) was overwritten by a "Z"!
- If null overwritten, c-string no longer "acts" like c-string!
  - Unpredictable results!

Library

- Declaring c-strings
  - Requires no C library
  - Built into standard C

- Manipulations
  - Require library: #include <string.h>
  - Typically included when using c-strings
    - Normally want to do "fun" things with them…

- <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))
```

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!