CSC180: Lecture 23

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

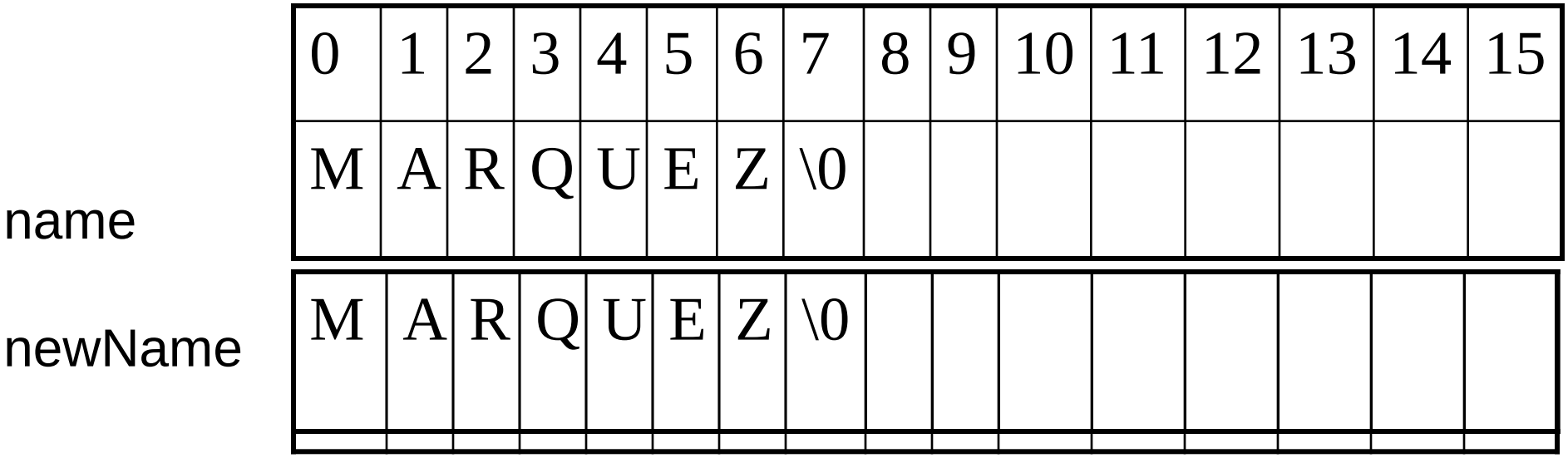
Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

= 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!

strcpy demo



strcpy(newName,name);

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

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

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))

strcmp - Comparison Function

```
const int NAME_SIZE = 16;
char name[NAME_SIZE], newName[NAME_SIZE];

if (name == newName)              //illegal, == not defined

int strcmp(s1,s2);
   if s1 = s2, 0 is returned            (false!)
   if s1 < s2, negative is returned (true)
   if s1 > s2, positive is returned   (true)
```

strcmp Usage

```
if (strcmp(name, newName))
    true is the not equal condition (non-zero)
else
    false is the equal condition       (zero)

//Better
if (strcmp(name, newName) == 0)
    true is the equal condition
else
    false is the not equal condition
```

strcmp demo

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
name	M	A	R	Q	U	E	Z	\0								
newName	M	A	R	Q	U	E	S	\0								

strcmp(name, newName)

Returns 7.

stricmp – ignore case comparison

```
const int NAME_SIZE = 16;
char name[NAME_SIZE] = “Marquez”;
char newName[NAME_SIZE] = “ MARQUEZ”;

if (stricmp(name, newName) == 0)
    true – comparison would be true
else
    false
```

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!

C-string Functions: strcat()

```
char* strcat( char *s1, const char *s2 )
```

- Appends second argument to first argument
  - First character of second argument replaces null character terminating first argument
  - You must ensure first argument large enough to store concatenated result and null character

-
```
char* strncat( char *s1,
               const char *s2,
               size_t n )
```
  - Appends specified number of characters from second argument to first argument
    - Appends terminating null character to result

- <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++.

`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++.

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

Functions working on C-Strings

isalnum	**toupper**	**strchr**
isalpha	**atoi**	**strcspn**
iscntrl	**atof**	**strpbrk**
isdigit	**atol**	**strrchr**
isgraph	**strtod**	**strspn**
islower	**strtol**	**strstr**
isprint	**strtoul**	**strcmp**
ispunct	**strcat**	**strncmp**
isspace	**strcpy**	**strlen**
isxdigit	**strncpy**	**strerror**
tolower	**strtok**	

C-String functions: sscanf and sprintf

- Syntax :

  sscanf (instring, “format”, variable_address);

  - Similar to scanf, it inputs data from string instead of keyboard
  - Format: similar to those used in printf, see Chapter 4
  - If use %s format, it reads all until it reaches the first white space

- Syntax:

  sprintf (outstring, “format”, variable);

  - Similar to printf, it outputs results to string instead of screen

C-String functions: sscanf and sprintf

```
char     c;
int      i;
double db;
char     s[80], outstring[80],
         *instring="a 100 -1.23 This is a sample data";

sscanf(instring,"%c%d%lf%s", &c, &i, &db, s);
//Then c='a', i=100, db=-1.23, s="This"

sprintf(outstring,"%c%d%lf%s", c, i, db, s);
// print to outstring: a 100 -1.23 This
```