# iClicker Attendance

Please click on A if you are here:

A. I am here today.

# Strings in C

# What is a "string"?

- A "string" is just a vector of ASCII characters
  - Followed by a "null terminator" – a byte with the value 0 or 0x00
  - Warning: blanks in ASCII have a non-zero value (0x20)

```
char str[14]="This a string";
```

{'T', 'h', 'i', 's', ' ', 'a', ' ', 's', 't' ,'r' ,'i' ,'n' ,'g', x00}

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ASCII | T | h | i | s | | a | | s | t | r | i | n | g | |
| Hex | x54 | x68 | x69 | x73 | x20 | x61 | x20 | x73 | x74 | x72 | x69 | x6e | x67 | x00 |

# The “null terminator”

- A null terminator is a single byte with the value 0b00000000
  - A null terminator is NOT a null address… not NULL! 0b00000000_00000000_00000000_00000000
  - A null terminator is NOT an *integer* 0 0b00000000_00000000_00000000_00000000
- When we assign a word 0 to a char 0, the left bits are truncated
  - So we can assign 0 to a char to get a null terminator
- When we compare a null-terminator to a word 0, the null terminator is extended to word length
  - So we can compare a byte to 0 to see if it is a null terminator
- To avoid confusion, I use 0x00 as a null terminator (0b00000000)
- You will also see ‘\0’ – the ASCII escaped null terminator (0b00000000)

# Using Pointer Notation

- We already learned that C lets us use pointer notation for arrays

```
int * c3; // is very similar to int c3[];
```

- When dealing with strings, we almost always use pointer notation
  - Length is not important BECAUSE we have a null terminator

```
char * string; // is very similar to char string[];
```

# Pointer notation and Memory

- Pointer notation does NOT reserve space for a string!

```
char * string;
string[0]=0x00; // results in SEGMENTATION VIOLATION!
```

- A literal string DOES reserve space for the literal string

```
char * string="This is a test"; // string points to literal
string[0]='X'; // Modify the literal in memory*
               // (May cause segmentation violation!)
printf("String is: %s\n",string); // String is: Xhis is a test
```

# printf substitutes string for %s

```
char str[14]="This a string";
printf("Variable str contains |%s| and no more\n",str);
```

Variable str contains |This a string| and no more

- %s replaced by characters…
  - starting at pointer-to-character argument
  - up to a null terminator

# Empty String

```
char str[14]="This a string";
str[0]=x00;
printf("Variable str contains |%s| and no more\n",str);
```

Variable str contains || and no more

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ASCII | | h | i | s | | a | | s | t | r | i | n | g | |
| Hex | x00 | x68 | x69 | x73 | x20 | x61 | x20 | x73 | x74 | x72 | x69 | x6e | x67 | x00 |

# Standard Library String Functions

```
#include <string.h>
char str[18]="This a string";
printf("Size of str buffer: %d\n",sizeof(str));
printf("Length of str string: %d\n",strlen(str));
```

Size of str buffer: 18

Length of str string: 13

| Indx | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| char | T | h | i | s | | a | | s | t | r | i | n | g | \Ø | \Ø | \Ø | \Ø | \Ø |
| Hex | x54 | x68 | x69 | x73 | x20 | x61 | x20 | x73 | x74 | x72 | x69 | x6e | x67 | x00 | x00 | x00 | x00 | x00 |

# iClicker Question

- What does this line of C code do?

```
string[strlen(string)]=0x00;
```

A. Adds null terminator to the end of string
B. Wastes time and does nothing
C. Shortens the length of string by 1 character
D. Puts a second null terminator after the first
E. None of the above

# strlen(char * str);

- Counts the number of characters up to (but not including) the null terminator of the argument.

- Because we start indexes at zero:

```
str[strlen(str)]==0x00 // ALWAYS TRUE!
```

```
char empty[20]="";
printf("Length of empty: %d\n",strlen(empty));
```

Length of empty: 0

# strcpy(char * to,char * from)

- Copies “from” string to “to” string
- ASSUMES “to” is large enough to hold strlen(from)+1 characters
  - Including from’s null terminator

```
char buf[100]=“Old string“;
char new[20]=“Newer string”;
strcpy(buf,new);
printf(“Variable buf contains |%s| and no more\n”,buf);
```

Variable buf contains |Newer string| and no more

# strcat(char start[ ],char tail[ ])

- Copies “tail” string at the end of “start” string
- ASSUMES “start” is large enough to hold both “start” and “tail”
  - Including tail’s null terminator

```
char start[100]=“Beginning “;
char end[20]=“of a test.”;
strcat(start,end);
printf(“Variable start contains |%s| and no more\n”,start);
```

Variable start contains |Beginning of a test.| and no more

# strncat(char start[ ],char tail[ ],int n)

- Copies up to n bytes of "tail" string at the end of "start" string
  - If tail's null terminator is not copied, result is not null terminated!
- ASSUMES "start" is big enough to hold both start and n bytes of tail
- Safer than strcat if used correctly

```
char start[100]="Beginning "; // Note… short initialization!
char end[20]="of a test.";
strncat(start,end,6);
printf("Variable start contains |%s| and no more\n",start);
```

Variable start contains |Beginning of a t| and no more

# To be totally safe….

```
int bytesLeft=sizeof(start)-strlen(start)-1;
strncat(start,end,bytesLeft);
start[sizeof(start)-1]=0x00;
```

# strcmp(char * a, char * b)

- Compares the string in “a” to the string in “b”
  - If a<b, returns a number less than zero
  - If a==b, returns zero
  - If a>b, returns a number greater than zero
- Cannot compare strings with ==, <, >, <=, etc. operators!
  - Can compare CHARACTERS with ==, <, …

```
if (0==strcmp(name,”Tom”)) printf(“Hi Tom…”);
```

# strncmp(char *a, char *b,int n)

- Checks to see if a's prefix matches b's prefix for up to n characters
  - If a<b, returns a number less than zero
  - If a==b, returns zero
  - If a>b, returns a number greater than zero

```
if (0==strncmp(name,"Tom",3))
        printf("Name %s starts with Tom…",name);
```

Name Tompkins County starts with Tom

# Resources

- Programming in C, Chapter 6 (Arrays)
- Programming in C, Chapter 9 (Strings)
- Wikipedia C String Handling
  https://en.wikipedia.org/wiki/C_string_handling
- C String Tutorial :
  http://www.tutorialspoint.com/cprogramming/c_strings.htm