# CIS*1500 Lab 11

INTRODUCTION TO PROGRAMMING

## Announcements

- Assignment 2 due Sunday, November 29 at 11:55pm
- Lab Exam Two will be graded this week
- Solutions to previous lab exercises posted
- Next week lab
- Extra Help
  - Tues, Thurs, 2:30 – 4:30, 5:30 – 7:30 in the basement of Reynolds

- **Today's Lab**
  - Array Review
  - Strings and Pointers
  - Lab Exercise

## Passing Arrays Through Functions

- Arrays are passed to functions **by reference.**
- This means that the variable's address in memory is passed to the function.  As such, the variable you access in the function is not a copy of your variable in main, but instead the variable itself.
- Be careful if you declare an array inside of a function, the array will disappear when the function returns! Declare your arrays in main!
- Only the name of the array is passed to the function as argument or parameter.

```
arrayName
printArray(arrayName);
```

## Passing Arrays Through Functions

- Arrays as function arguments in C can be declared 3 ways:

```
void myFunction(char * param)  {
}

void myFunction(char param[]) {
}

void myFunction(char param[10]) {
}
```

# What is a String?

- **Strings** are actually one-dimensional array of characters terminated by a null character ‘\0’.

```
char string[6] = { ‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\o’};
char string[] = { ‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\o’};
```

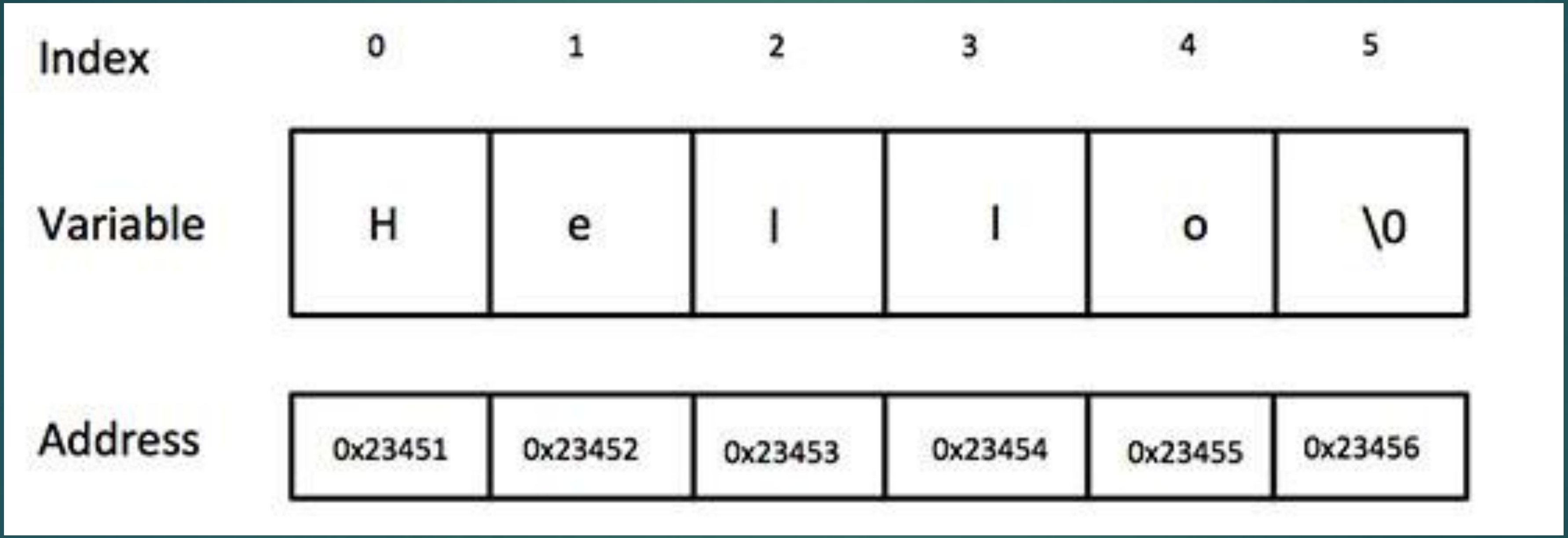
- This can be re-written as:

```
char string[] = “hello”;
char string[6] = “hello”;
char string[20] = “hello”;
```

- Strings are surrounded by double quotation marks whereas single characters are surrounded by single quotation marks.

# What is a String?

- Here is a memory presentation of the string defined earlier.



| Index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Variable | H | e | l | l | o | \0 |
| Address | 0x23451 | 0x23452 | 0x23453 | 0x23454 | 0x23455 | 0x23456 |

# The Null Terminator

- **The null terminator** is a special character, **‘\0’**, which in a character array specified the end of the string.
- At least one null terminator must be present in the character array for it to be considered a valid string.
- Use the null terminator in a for loop to **initialize your string**.
- In addition, all string functions check for the null terminator.

The string **“A”** is equivalent to the characters **‘A’** and **‘\0’**.

## Format Specifier for Strings

- Because it is really inconvenient to print out the contest of a character array one element at a time, a for specified exists for strings in the standard I/O library (stdio.h):

  **%s**

- Hence, we can print out the content of strings:

  **printf("Your string is: %s\n", myString);**

- **someString** is still a character array which is terminated by a null character.

## Scanning a String

- Similar functionality works with the **scanf** function using the string format specifier:

```
printf("Enter your string: ");
scanf("%s", myString);
```

- Why don't we need an ampersand **'&'** in front of **myString?**
- This is because myString, the name of the array is the pointer to the first element of the array.

## Scanning a String

- Be careful! If you enter:

  **Have a good afternoon**

```
printf("Enter your string: ");
scanf("%s", myString);
```

- Only "**Have**" will be scanned in. scanf only scans until the first whitespace character and then stops.
  - A space, a tab, or a new line.
- A way to bypass this is to use the **fgets** function.

# fgets Function

- The format of **fgets** is:

  **char * fgets(char * str, max_char, input_stream);**

- **str** – The pointer to an array of characters where the string read is stored.
- **max_chars** – The maximum number of characters to be read (including the null terminator).
- **input_stream** – The pointer to a FILE object that identifies the stream where characters are read from.

- On success, fgets returns the same **str** parameter. Otherwise, if the **End-of-File** is encountered and no characters have been read, the contents of **str** remain unchanged and a null pointer is returned.

## fgets Function

- **Be careful!** Whereas **scanf discards** the first whitespace character and does not include it in your string, **fgets includes** the newline character **'\n'** in the string!
- This means, that the last two characters of any string inputted with **fgets** are **'\n'** and **'\0'**.

- When main gets invoked, it already has three predefined streams open and available for use:
  - **FILE * stdin** – The standard input stream
  - **FILE * stdout** – The standard output stream
  - **FILE * stderr** – The standard error stream

## fgets Function

- So now if you enter:

**Have a good afternoon**

```
printf("Enter your string: ");
fgets(myString, 4096, stdin);
```

- Now, "**Have a good afternoon**" will be scanned in. Keep in mind it also scans in the newline and null terminator.

## Reading Strings From a File

- Download the file **scanFile.c** and **sample.txt** from bucky under Course Information -> Extra Resources -> Presentations and Resources from Labs.
- And follow along!

## The String Library

- Strings have many special functions associated with them and they can be accessed with the **string** library.

**#include <string.h>**

- The string library contains dozens of functions that can be used to manipulate and work with strings including:
  - Finding the length of a string
  - Copying one string to another
  - Concatenating strings
  - And more…

# strlen Function

- The function **strlen** returns the number of characters in the string **not** including the null terminator.
- The format of **strlen** is:

  **size_t strlen(const char *str)**

- **Str** – The string whose length is to be found
- On success the function returns the length of the string.

- **size_t** – Is an unsigned integer type meaning it cannot represent negative values.

## strlen Example

```
void function(char string[])
{
    int i;
    int length;
    i = 0;
    length  = strlen(string);

    for (i = 0; i < length; i++)
    {
        printf(“%c\n”, string[i]);
    }
}
```

## strcpy Function

- The function **strcpy** will allow you to copy the contents of one string into another.
- The format of **strcpy** is:

**char * strcpy(char * dest, const char * src)**

- **dest** – The pointer t the destination array where the content is to be copied.
- **src** – The string to be copied.
- On success returns a pointer to the destination string **dest.**

## strcat Function

- The function **strcat** is used to concatenate strings together. That is, it is used to insert the contents of one string to the end of the contents of another string. Also overwriting the null terminator **'\0'** of the first string.
- The format of **strcpy** is:

  **char *strcat(char *dest, const char *src)**

- **dest** – The pointer to the destination array, which should contain a C string, and should be large enough to contain the concatenated resulting string
- **src** – The string to be appended. This should not overlap the destination.
- On success, the function returns a pointer to the resulting string **dest**.

## Some More String Functions

- **int atoi(const char *str)**
  - Converts the string argument **str** to an integer (type int).
- **long int strtol(const char *str, char **endptr, int base)**
  - Converts the initial part of the string in **str** to a **long int** value according to the given base, which must be between 2 and 36 inclusive, or be the special value 0.
  - Use base = 10 for regular numbers.

# Some Character Functions

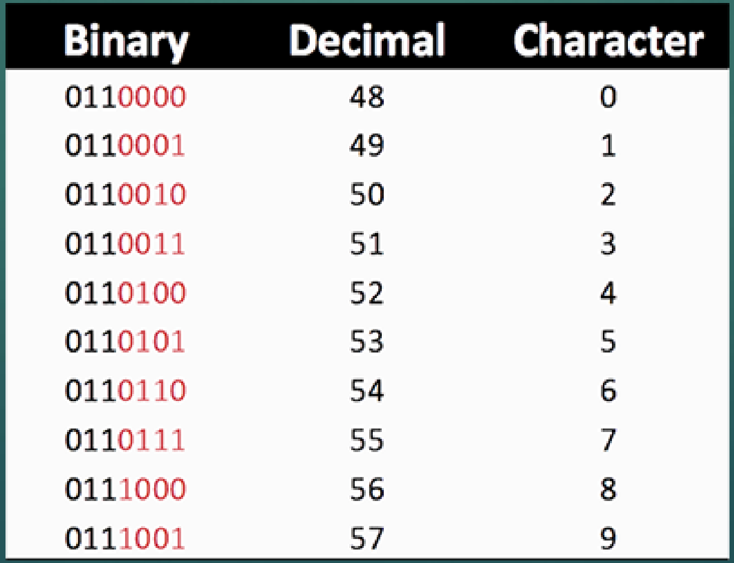
- These functions require the **character type** library:

  **#include <ctype.h>**

- **int isalpha(int c)**
  - Returns non-zero if c is an alphabet character, 0 otherwise.
- **int isupper(int c)**
  - Returns non-zero if c is an uppercase letter, 0 otherwise.
- **int islower(int c)**
  - Returns a non-zero if c is a lowercase letter, 0 otherwise
- **int isdigit(int c)**
  - Returns non-zero if c is a digit (0, 1, …, 9), 0 otherwise.

# Some Character Functions

- **NOTE:** These functions take a single argument in the form of an integer and return an integer value.
- Even though, they take an integer as an argument, **a character** is still passed to these function. When a character is passed as an argument, corresponding **ASCII Value** of that character is passed instead of that character itself.



| Binary | Decimal | Character |
|---|---|---|
| 0110000 | 48 | 0 |
| 0110001 | 49 | 1 |
| 0110010 | 50 | 2 |
| 0110011 | 51 | 3 |
| 0110100 | 52 | 4 |
| 0110101 | 53 | 5 |
| 0110110 | 54 | 6 |
| 0110111 | 55 | 7 |
| 0111000 | 56 | 8 |
| 0111001 | 57 | 9 |

## Lab Exercise

- Create a new folder for this lab exercise.
- Download the files **listProg.c** and **list.txt** from bucky under Course Information -> Extra Resources -> Presentations and Resources from Labs.
- Follow the instructions from the **listProg.c** to build the program.
- Push it onto git when you're done.

**Show your TA when you're done**

**Solution will be posted at the end of this week**