COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Week 11 - Class II: Strings Part II

Administrivia

- SPA 3 now due on Fri.
- SPA 4 and LPA 2 will come out today, will be due on November, 8th, and November 15th respectively.
- Exam grades will be released today.
- Mid-Semester Feedback Survey will be posted today.
  - Please complete to count as a quiz grade.

Today's Agenda

1. Quick Recap of last Class
2. String Library Functions!

Review

Reading Input into Arrays (Wrong!)

```c
int num[2];
int num2[2];
int mynum[2];

printf("Enter: ");
scanf("%d", num);
printf("Enter: ");
scanf("%d", num2);
printf("Enter: ");
scanf("%d", mynum);

for(int i = 0; i < 2; i++){

printf("num[%d] = %d\n", i, num[i]);
printf("num2[%d] = %d\n", i, num2[i]);
printf("mynum[%d] = %d\n", i, mynum[i]);

}
```

This will result in garbage being saved to the array after each first slot.

Collecting a String with `scanf`

- Collecting input for a string follows very similar procedures as collecting other data types.
- Two Differences:
  - Placeholder %s
  - No address operator (&)

Why no address operator needed?

```
char pokemon[10];

scanf('%s', pokemon);

printf("the pokemon is %s\n", pokemon);
```

Collecting a String with `scanf`

- Collecting input for a string follows very similar procedures as collecting other data types.
- Two Differences:
  - Placeholder %s
  - No address operator (&)

Remember the array variable is a pointer to the 1st adjacent memory celling the stack space.

```c
char pokemon[10];

scanf('%s', pokemon);

printf("the pokemon is %s\n", pokemon);
```

Limitations of using `scanf` for Strings

- Strings are just an array characters.
- We can technically form words and even phrases.
- Does a single `scanf` statement with ONE placeholder allow multiple words to be collected into some string?

```c
char pokemon[10];

scanf('%s', pokemon);

printf("the pokemon is %s\n", pokemon);
```

Limitations of using `scanf` for Strings

- Strings are just an array characters.
- We can technically form words and even phrases.
- Does a single `scanf` statement with ONE placeholder allow multiple words to be collected into some string?

```
char phrase[30];

scanf('%s', phrase);

printf("the phrase is %s\n", phrase);
```

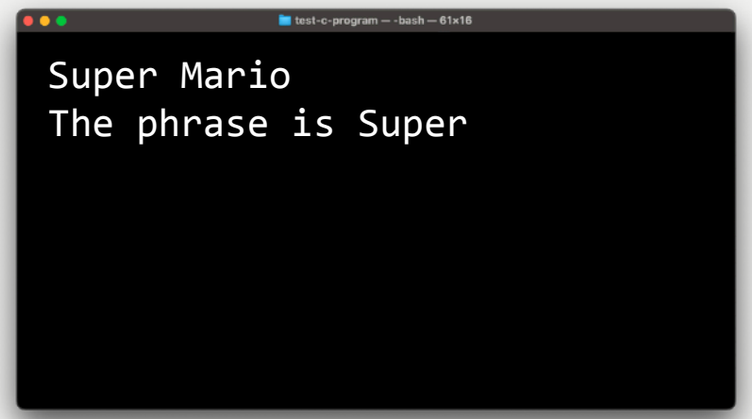
What if we type “Super Mario”?

Limitations of using `scanf` for Strings

- Strings are just an array characters.
- We can technically form words and even phrases.
- Does a single `scanf` statement with ONE placeholder allow multiple words to be collected into some string?

```c
char phrase[30];

scanf('%s', phrase);

printf("the phrase is %s\n", phrase);
```

What if we type “Super Mario”?



Limitations of using `scanf` for Strings

- Strings are just an array characters

Scanf stops reading values for a string when it encounters the whitespace character (“ “)!

Reading Strings into Arrays

```c
char word[8];
printf("Enter: ");
scanf("%s", word);
printf("word = %s\n", word);
```

Reading Strings into Arrays

```
char word[8];
printf("Enter: ");
scanf("%s", word);
printf("word = %s\n", word);
```

Why aren’t garbage values being displayed?

The null character!

Limitations of using `scanf` for Strings

- C (and even other programming languages) has a buffer input stream.
- A stream is a sequence of characters.
  - Think of it as like a flowing river of incoming characters.
- Buffer is a temporary storage area that stores characters.
- It's very important to understand that the buffer must always be empty (meaning no characters are stored) in order to properly collect input.
- When `scanf` is used to collect a string, any other character including the space is still in the buffer.

More Strings

`getchar()` Function

- **`getchar`** is a function in C that retrieves a character from the buffer standard input stream (stdin).
- The function returns the integer ascii value of the respective character.
- If the standard input stream is empty, then it is basically the equivalent of using a **`scanf`** statement that reads a single character.

```c
char ltr = getchar();
```

Clearing the Input Buffer

- Here is a simple user defined function that you can use to clear the input buffer.

- This function should be only when the buffer needs to be cleared. In other words, if your code is skipping an input collection statement, then that means the buffer had readable content.

```
void clearBuffer(){
    while(getchar() != '\n');
}
```

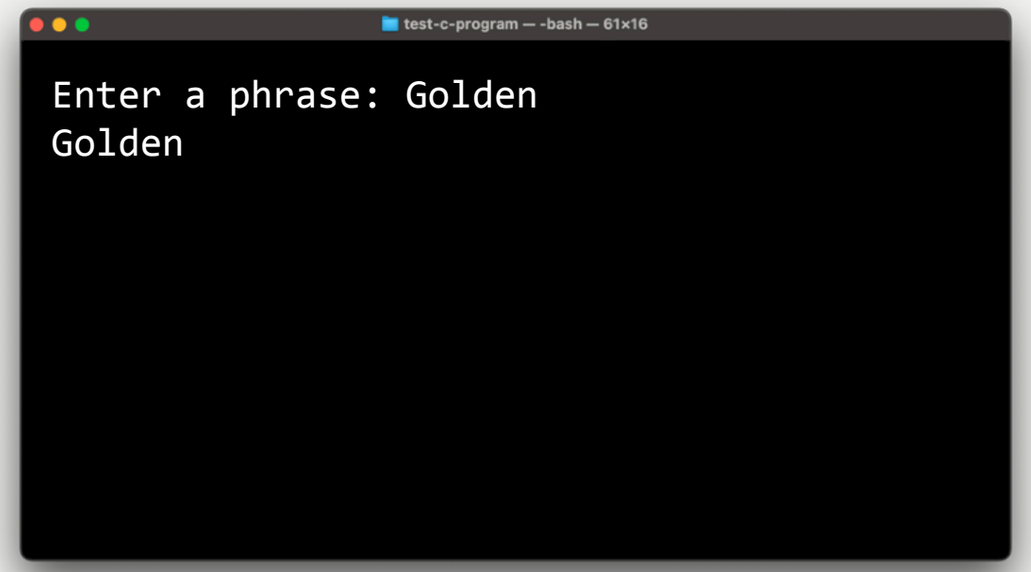
`gets()` and `puts()`

- Something we've noticed is that scanf has some limitations.
- scanf only allows one word to be read.
- How can programmers input a sentence as string?
- `gets()` is a simple function that allows user to input more than one word that can be stored in a character array (allowing whitespaces).
- `puts()` is another way to display a string onto the screen.

`gets()` and `puts()` Example

```c
char phrase[10];
printf("Enter a phrase: ");
gets(phrase);
puts(phrase);
```

```
'gets' has been explicitly marked
deprecated here
__deprecated_msg("This function is
provided for compatibility reasons only.
Due to security concerns inherent in the
design of gets(3), it is highly
recommended that you use fgets(3)
instead.")
```

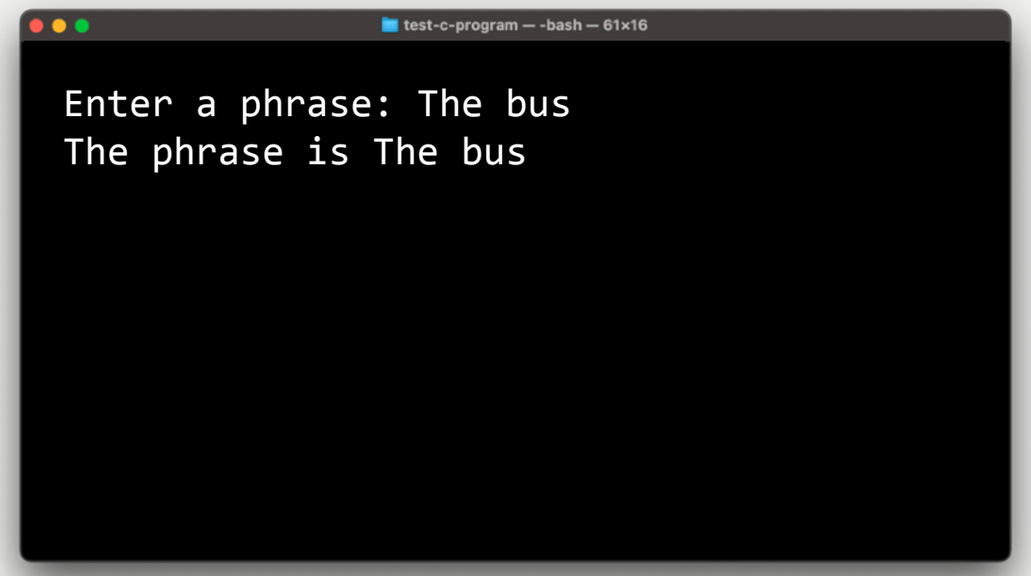


`fgets()`

- `fgets()` is similar to gets(), but with extra syntax.
- `fgets()` meets the possible that gets() raises.
- `fgets()` takes three arguments
  - Array
  - String Length Limit
  - File to read from (stdin which is standard input)
- **`fputs()`** works like **`puts()`**, except that it doesn't automatically append a newline

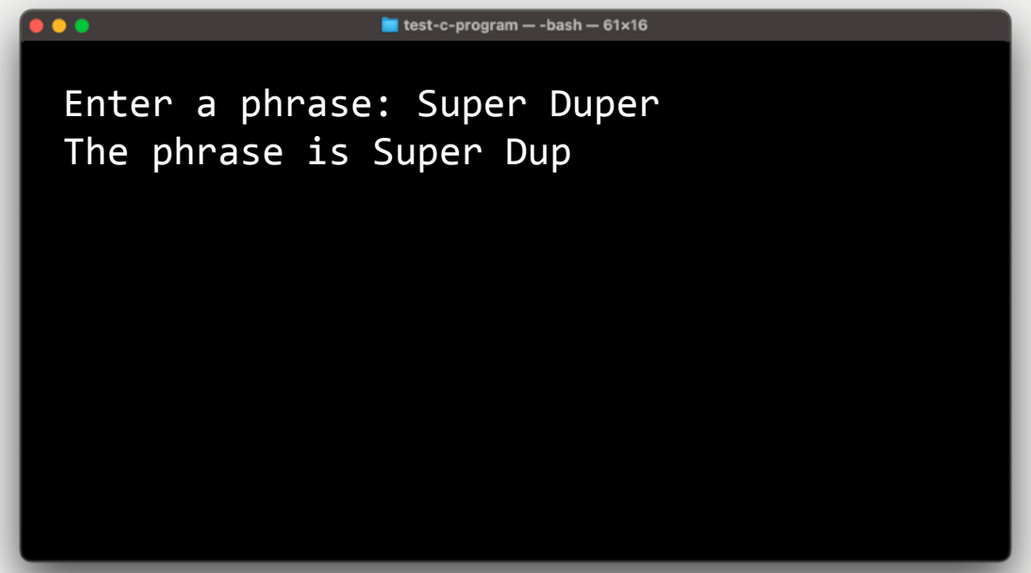
`fgets()` Example

```c
char phrase[10];
printf("Enter a phrase: ");
fgets(phrase, 10, stdin);

printf("The phrase is %s\n", phrase);
```



`fgets()` Example

```c
char phrase[10];
printf("Enter a phrase: ");
fgets(phrase, 10, stdin);

printf("The phrase is %s\n", phrase);
```



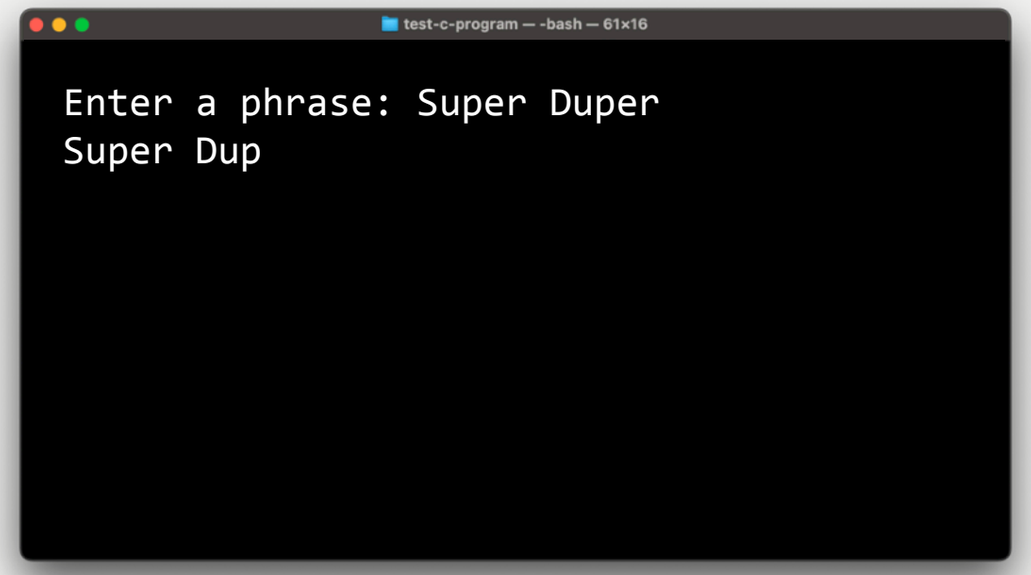
Understanding how `fgets()` works

- `fgets` is a function that allows to process characters (including a whitespace characters) until newline character (‘\n’) is read

- If `fgets` receives a strings bigger than the provided limit, it will just append the null character in the last element and send the remaining characters into the buffer space

`fputs()` Example

```c
char phrase[10];
printf("Enter a phrase: ");
fgets(phrase, 10, stdin);

fputs(phrase, stdout);
```



The String library

- Strings has a library devoted to strings.
- The library contains a series of functions that can manipulate or access certain content about strings.
- All functions associated with strings are stored in the string header file (**`string.h`**)
- Since they are stored in separate header file, make sure to include it!!

```
#include<string.h>
```

The String library

Function	Stack Space
`strcpy()`	Makes a copy of source, a string, in the character array accessed by dest:
`strncpy()`	Makes a copy of up to n characters from source in dest: strncpy(dest, source, 5) stores the first five characters of the source and does NOT add a null character.
`strcat()`	Appends source to the end of dest: strcat(dest, source)
`strncat()`	Appends up to n characters of source to end of dest, adding the null character if necessary.
`strcmp()`	Compares s1 and s2 alphabetically. Returns a negative value if s1 should precede s2, a zero if strings are equal, and a positive value if s2 should precede s1 in an alphabetized list. strcmp(s1,s2)
`strncmp()`	Compares the first n characters in s1 and s2 returning positive, zero, and negative values like strcmp.
`strlen()`	Returns the number of characters in s, not counting the terminating null. strlen(s)
`strtok()`	Breaks the parameter string into tokens finding groups of characters separated by any of the delimiter characters. Each group is separated with '\0'.
`strchr()`	Returns a pointer to the first location of a character located in the string. Null is returned if character is not found.
`strpbrk()`	Return a pointer to the first location in the strings that holds any character found in another string.
`strchr()`	Returns a pointer to the last occurrence of a character in the string. Null is returned if character not found.
`strstr()`	Returns a pointer to the first occurrence of string s2 in string s1. Null is returned if character not found.

Demo

Acknowledgements

Slides adapted from Dr. Andrew Steinberg's COP 3223H course