COMP 206 – Intro to Software Systems

Lecture 9 – <string.h>, text input/output, debugging

September 28th, 2018

Warm-up and Recall: Word counting with pointers

```c
#include <stdio.h>

int main()
{
        char string[100] = "The quick brown fox jumped over the pile of ice.";

        char *pos = string;
        unsigned int words = 0;
        while(*pos){                                    // Note that the while condition becomes false when we hit \0
                if( *pos  == ' ' || *pos == '.' ){
                        words++;
                }

                printf( "Processing character %c with words %d.\n", *pos, words );

                pos++;                                  // Pointer addition moves us forward by 1 char in memory
        }

        printf( "There were %d words.\n", words );

        return 0;
}
```

C Built-in Libraries

- It's important that we know how to do operations "bit-by-bit"
  - We are just at the byte-by-byte stage for now, just wait a few weeks!
  - This helps us to understand what's really happening in a system

- Once we get it, then we want to build bigger programs quickly. Standardized implementations of basic operations are provided as language libraries:
  - <stdio.h>
  - <stdlib.h>
  - <string.h>
  - <math.h>
  - etc

Key functions within <string.h>

size_t strlen(const char *str) Computes the length of the string str up to but not including the terminating null character.	**void *memset(void *str, int c, size_t n)** Copies the character c (an unsigned char) to the first n characters of the string pointed to, by the argument *str*.
int strcmp(const char *str1, const char *str2) Compares the string pointed to, by *str1* to the string pointed to by *str2*.	**char *strcat(char *dest, const char *src)** Appends the string pointed to, by *src* to the end of the string pointed to by *dest*.
char *strcpy(char *dest, const char *src) Copies the string pointed to, by *src* to *dest*.	**char *strstr(const char *haystack, const char *needle)** Finds the first occurrence of the entire string *needle* (not including the terminating null character) which appears in the string *haystack*.

C library function - strcpy()

Previous Page Next Page

Description

The C library function **char *strcpy(char *dest, const char *src)** copies the string pointed to, by **src** to **dest**.

Declaration

Following is the declaration for strcpy() function.

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

Parameters

- **dest** – This is the pointer to the destination array where the content is to be copied.
- **src** – This is the string to be copied.

Return Value

This returns a pointer to the destination string dest.

Example

The following example shows the usage of strcpy() function.

Live Demo

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

int main () {
   char src[40];
   char dest[100];

   memset(dest, '\0', sizeof(dest));
   strcpy(src, "This is tutorialspoint.com");
   strcpy(dest, src);

   printf("Final copied string : %s\n", dest);

   return(0);
}
```

Let us compile and run the above program that will produce the following result –

```
Final copied string : This is tutorialspoint.com
```

C library function - strstr()

Previous Page Next Page

Description

The C library function **char *strstr(const char *haystack, const char *needle)** function finds the first occurrence of the substring **needle** in the string **haystack**. The terminating '\0' characters are not compared.

Declaration

Following is the declaration for strstr() function.

```
char *strstr(const char *haystack, const char *needle)
```

Parameters

- **haystack** – This is the main C string to be scanned.
- **needle** – This is the small string to be searched with-in haystack string.

Return Value

This function returns a pointer to the first occurrence in haystack of any of the entire sequence of characters specified in needle, or a null pointer if the sequence is not present in haystack.

Example

The following example shows the usage of strstr() function.

Live Demo

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


int main () {
   const char haystack[20] = "TutorialsPoint";
   const char needle[10] = "Point";
   char *ret;

   ret = strstr(haystack, needle);

   printf("The substring is: %s\n", ret);

   return(0);
}
```

Let us compile and run the above program that will produce the following result –

```
The substring is: Point
```

Key functions within <stdio.h>

FILE *fopen(const char *filename, const char *mode) Opens the filename pointed to by filename using the given mode.	**int fclose(FILE *stream)** Closes the stream. All buffers are flushed.
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream) Reads data from the given stream into the array pointed to by ptr.	**size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)** Writes data from the array pointed to by ptr to the given stream.
long int ftell(FILE *stream) Returns the current file position of the given stream.	**int fseek(FILE *stream, long int offset, int whence)** Sets the file position of the stream to the given offset. The argument *offset* signifies the number of bytes to seek from the given *whence* position.

Deeper Look: fopen(...)

- Returns a "file pointer":
  - NULL (=0) if there was any problem -> always check for this!
  - Otherwise, it's safe to use the other file operations
- The mode string indicates what we want to do, so fopen can check that we have the correct permissions:
  - "r": read only. The file must exist previously with read permission.
  - "w": write only. Create new file or overwrite previous contents.
  - "a": append. Create new file or add to the end of previous contents.
  - "b" can be added to any (e.g., "rb"), meaning to intpret the file as binary. This is for next week.
  - "+" is a more complex idea saying we want to read and write together. This is not covered in 206 and I recommend you ignore it until you understand the basics, but you are free to explore as you wish. It's ok for assignments if you use it correctly.

Example: How large is the file?

```
FILE* fp = fopen( "myfile.txt", "r" );
size_t sz;
fseek(fp, 0L, SEEK_END);
sz = ftell(fp);
rewind(fp);

char file_data_array[sz+1];    // One more for the \0
fread( file_data_array, 1, sz+1, fp );
printf( "File contents:\n%s\n", file_data_array );
```

Question: Does fread give you a \0 automagically?

- Good check for yourself: you should be able to test this by writing your own C program to process array after it's been returned from fread.
  - Pause here and try to answer yourself. What would you do?

Question: Does fread give you a \0 automagically?

- Good check for yourself: you should be able to test this by writing your own C program to process array after it's been returned from fread.
  - Pause here and try to answer yourself. What would you do?

Dave's answer: run this loop and see if "0" is the last thing printed.

```
for( int pos=0; pos<sz+1; pos++ ){
    printf( "Character %d has AASCI value %d.\n", pos, array[pos] );
}
```

More key functions within <stdio.h>

int printf(const char *format, ...) Sends formatted output to stdout.	**int fprintf(FILE *stream, const char *format, ...)** Sends formatted output to a stream.
int fputs(const char *str, FILE *stream) Writes a string to the specified stream up to but not including the null character.	**char *fgets(char *str, int n, FILE *stream)** Reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.
int fputc(int char, FILE *stream) Writes a character (an unsigned char) specified by the argument char to the specified stream and advances the position indicator for the stream.	**int fgetc(FILE *stream)** Gets the next character (an unsigned char) from the specified stream and advances the position indicator for the stream.

Discussion: fgetc, why return int?

- We want to use fgetc in a loop, perhaps to read everything:

```
while( input_char = fgetc( fp ) ){
    printf( "I read the character %c.\n", input_char );
}
```

Discussion: fgetc, why return int?

- We want to use fgetc in a loop, perhaps to read everything:

```
while( input_char = fgetc( fp ) ){
    printf( "I read the character %c.\n", input_char );
}
```

THIS IS AN ERROR:
Do not copy!

Discussion: fgetc, why return int?

- We want to use fgetc in a loop, perhaps to read everything:

```
while( (input_char = fgetc( fp )) != EOF ){
    printf( "I read the character %c.\n", input_char );
}
```

This works!

- To do this, we need some special character that could never exist in the text itself.
- EOF has the value –1. This is not a valid AASCI code, so we cannot mistake it for real data.
    - Special meaning: once you read EOF, must stop grabbing from the file.

Text file formats

- Can be unstructured, for the purposes of human consumption:
  - E.g., the text version of a novel, your journal, written answers
  - Common extension: ".txt"
- More interesting for systems: structured text files
  - E.g., A C program, a BASH program, a website on the internet, and lots more
- Thinking text as data for our computation:
  - Goal: allow programs to easily save and restore content, not necessarily easy to read for humans, but also a "nice to have"
  - Systems programmers are responsible for creating good file types. How to split up the data, tell where one item ends and another begins, make processing fast

Text data file examples

- CSV – Comma separated values
- HTML – Hyper-text markup language
- MD – Markdown
- YAML – Yet another markup language
- JSON – Javascript Object Notation

The contents of a CSV (the names of the next hurricanes?)

```
2019, alice, bob, clark, david, edward
2020, angus, burt, clyde, daphne, emily
2021, alf, ben, cindy, debra, elmira
```

- Think about the C operations that would be required to:
  - read a file "hurricane_data.txt" from disk
  - extract each word into an individual C variable of type char[]
  - print the words one by one to terminal (each time a hurricane occurs)

Solutions on Github: ExampleCode/Lecture9-libraries_debug

- Good to look at these and try to replicate yourself. They practice the 3 different "styles" of working with text data we've seen:

- Using C's underlying "array of characters" data type:
  - parse_csv_arrays.c
- Using C pointers to keep addresses within the array:
  - parse_csv_ptrs.c
- Using the library functions from <string.h>
  - parse_csv_stringH.c

Tricks for Debugging – May help with A2!

```
#ifdef DEBUG
printf( "Var 1 is %s, Var 2 is %s, Var 3 is %s.\n", var1, var2, var3 );
#endif
```

- When compiled as "$ gcc –DDEBUG q1a_simple_diamond.c", this does the printing
- When compiled normally as "$ gcc q1a_simple_diamond.c", nothing is printed

Tricks for Debugging – May help with A2!

- Try the "gdb" debugger.
  - Compile with the "debug symbols" flag, "$ gcc –g q1a_simple_diamond.c"
  - Run "$ ./a.out", you will see the debugger open
  - Type "run 9" for example to run the code with argument 9 (give any arguments needed after run, just as you would have done after ./a.out)
- If your program ends abnormally (segfault, bus error, aborted), gdb will hold some info about what happened
  - "backtrace" shows line number that the program was on, which functions have been called in what order
  - "print varname" lets you inspect the value of variables at the crashed state
- You can also walk through your program from start onwards:
  - "break main" -> says to stop at the beginning
  - "run"
  - "next", "next", "next" -> runs one line at a time

Recommended Reading

- Browse the full descriptions of <string.h> and <stdio.h>
  - https://www.tutorialspoint.com/c_standard_library/stdio_h.htm
  - https://www.tutorialspoint.com/c_standard_library/string_h.htm

Exercises, more string operations

- Do these with basic array and pointer operations:
  - Copy one string into another:
    - Assuming char string1[] = “Hello”; char string2[] = “World”; Make string1 hold “World”.
    - Note that string1=string2 does not count here! Why?
  - Remove the ugly fourth word of the sentence:
    - char sentence[] = “COMP 206 is almost my favorite course.”
  - Capitalize the first letter of every word