CS 214: System Programming

(Sec 02 10:35 AM - 11:30 AM ARC-105)

Zhe(Jay)Wang
10/05/2017
Email: jay.wang@rutgers.edu
Office Hours: Friday 2:30pm-4:30pm at **Core 632**
(public space on the 6th floor next to elevator)

What will we do in this recitation?

- Review the HW last time
- Use C to implement the ‘ls’ command
- HW

warm up

How to debug（ project case）

Algorithm

small test case

Implementation

always
check input before exception

programming syntax

binary search

being humble…

What are the differences between strlen and sizeof a string in C? Why?

```c
#include "stdio.h"
#include "string.h"

int main()
{
    char str[] = "october"; // october is 7 letters
    printf("strlen %d\n",strlen(str));
    printf("sizeof %d\n",sizeof(str));
    return 0;
}
```

strlen 7
sizeof 8

What are the differences between strlen and sizeof a string in C? Why?

	sizeof	strlen
what is it	operator	function
when to be computed	compiling	running
parameter	array,pointer,type,function, struct	only char*
need to notice	can’t use for dynamic allocated memory	length not include ‘\0’

previous HW

1. Write some code that declares two arrays of size 10 that are string literals.

Make a pointer to one of the arrays, cast it to be an int pointer, and print out its value.

Make a new integer, set it equal to the value of your int pointer, then make a pointer to that integer, cast it to be a char pointer, and print out 8 chars.

What happened? Why?

2. Write some code that declares two arrays of size 10 that are string literals.
Create a pointer that points to the beginning of the first array, then in a loop, increment the pointer and print out the char it points to, out to index 20.
What happened? Why?

correctness for recitation today: String literal in c

2

There is no difference in how a **string literal** is stored in memory, regardless of whether or not you use the `constant` or `static` **storage class modifiers**, or if you use it as a function parameters as opposed to an intermediate variable/pointer: they are always stored in the **code segment**. The optimizer will also replace references to your intermediate pointer, `MY_STRING`, with the address to the literal itself.

Examples are shown below:

```
Example:                        Allocation Type:   Read/Write:  Storage Location:
=============================================================================
const char* str = "Stack";      Static             Read-only    Code segment
char* str = "Stack";            Static             Read-only    Code segment
char* str = malloc(...);        Dynamic            Read-write   Heap
char str[] = "Stack";           Static             Read-write   Stack
char strGlobal[10] = "Global"; Static              Read-write   Data Segment (R/W)
```

References

1. *Difference between declared string and allocated string*, Accessed 2014-07-31, <https://stackoverflow.com/questions/16021454/difference-between-declared-string-and-allocated-string>

refer
https://stackoverflow.com/questions/25068639/memory-usage-of-literal-strings-in-c
https://stackoverflow.com/questions/16021454/difference-between-declared-string-and-allocated-string

```
79    */
80        int i;
81        char stra[20] ="abcdefghvvv";
82        char strb[15]="abcdefghijklm";
83        char strc[10]="123456789";
84        char *p4=strb;
85        printf("--------------------------\n");
86        for (i=0;i<30;i++){
87           printf("the char index (%d) value (%c)\n",i,*p4);
88           p4++;
89        }
90
91        printf("addr of stra %p strb %p strc %p\n",stra,strb,strc);
92
93    }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 7: bash

```
the char index (25) value (v)
the char index (26) value ()
the char index (27) value ()
the char index (28) value ()
the char index (29) value ()
addr of stra 0x7fff5b0fc990 strb 0x7fff5b0fc981 strc 0x7fff5b0fc977
dessen@nwk-144-246 214 demo$
```

```c
    int i;
    char stra[20] ="abcdefghvvv";
    char strb[15]="abcdefghijklm";
    char strc[10]="123456789";
    char *p4=strb;
    printf("-------------------------\n");
    for (i=0;i<30;i++){
       printf("the char index (%d) value (%c)\n",i,*p4);
       p4++;
    }
    char *strd="abcd";

    char stre[]="Global";
    char strf[10]="abcdef";

    printf("addr of strd %p\n",strd);

    printf("addr of stra %p\n", stra);
    printf("addr of strb %p\n", strb);
    printf("addr of strc %p\n", strc);
    printf("addr of strf %p\n",strf);
    printf("addr of stre %p\n",stre);

}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 7: bash

```
the char index (24) value (v)
the char index (25) value (v)
the char index (26) value ()
the char index (27) value ()
the char index (28) value ()
the char index (29) value ()
addr of strd 0x107a80f24
addr of stra 0x7fff5817f990
addr of strb 0x7fff5817f981
addr of strc 0x7fff5817f977
addr of strf 0x7fff5817f96d
addr of stre 0x7fff5817f951
```

from low addr to high addr

code segment(strd)

stack(stra strb strc strf)

Data segment(stre)

1. Write the function

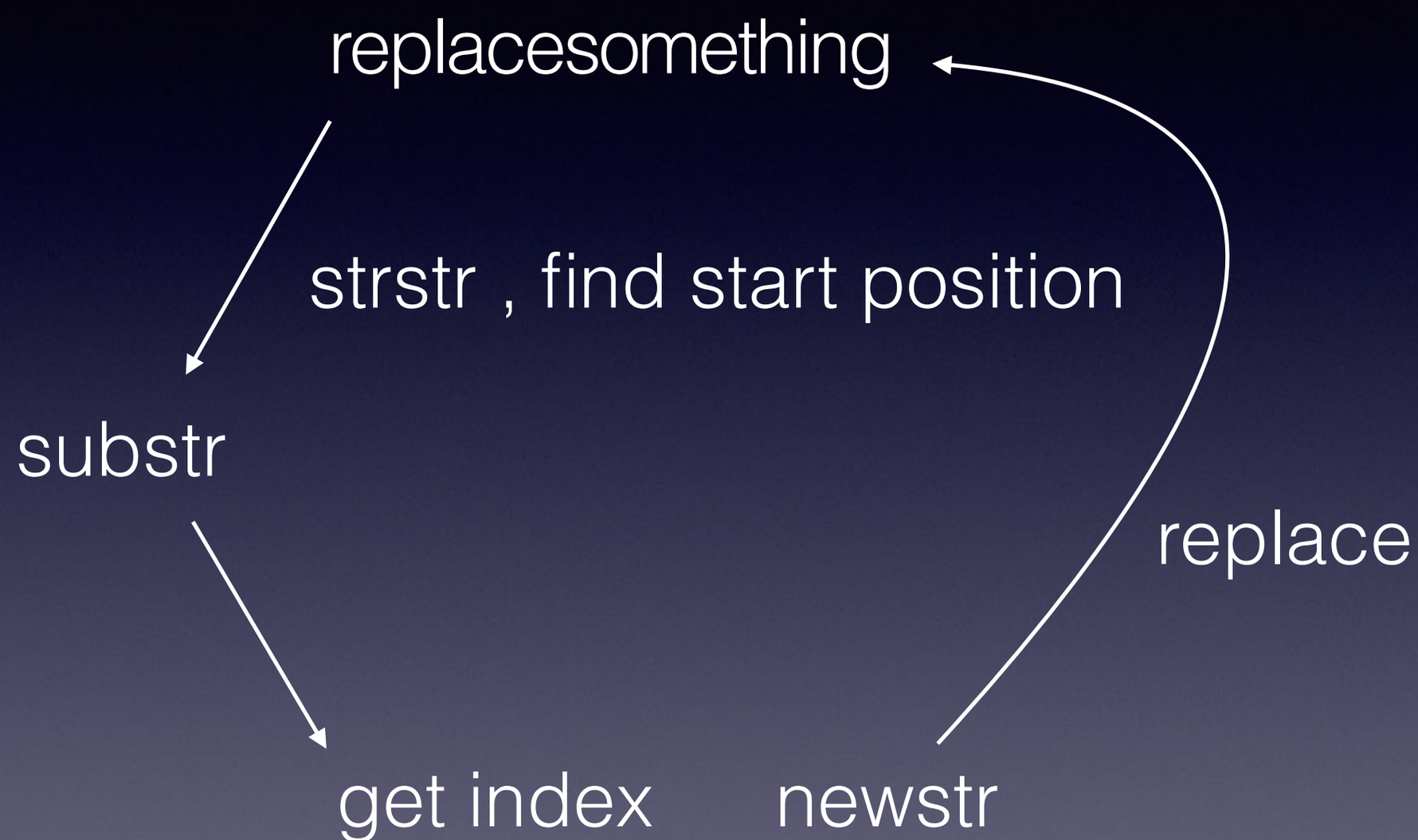
```
replace(char string[], char from[], char to[])
```

which finds the string from in the string string and replaces it with the string to. You may assume that from and to are the same length.

For example, the code

```
char string[] = "recieve";
replace(string, "ie", "ei");
```

should change string to "receive".



Demo, Knowing the position of the memory is important!

HW5: (implement ls)

0. Using opendir and readdir, open the current directory and output all filenames until there are no more char *

```
base = “./";
DIR * thingy = opendir(base);
dirent * newfile = readdir(thingy);
```

1. Parse the dirent struct to see if an entry is a directory or a file. If it is a file, prepend "./" to the filename, if it is a directory, don't.

```
…
if newfile != NULL
//check type field of newfile dirent struct to determine the type of this file
//endpoint
newfile->d_type
// compare with system defines for different endpoint types (3rd
//paragraph under 'NOTES' in man 3 readdir).
... if == DT_REG //regular file
elseif == DT_DIR //directory
....
```

2. Open a file handle to each file and use lseek to determine the file's size in bytes, and print out the file's size next to its name.

//assemble name of file using base directory and current path/name // concatenate all path up until now...
strcat(newpath, base)
// add currend name if it is a file...
newerpath = realloc(newpath, strlen(newpath)+strlen(newfile->d_name));
// d_name is REQUIRED to have a terminating null byte by standard ... yippee!
strcat(newerpath, newfile->d_name);

int checkFD = open(newerpath, RD_ONLY);
 ... if no error...
int len = lseek(checkFD, 0, SEEK_END);
close(checkFD);

printf(filename with full path, either color to indicate file/dir or put a "/" at the end to indicate dir, and number of bytes of size, if a file)

//be sure to closedir() when done with dir descriptor

unistd.h
is the name of the header file that provides access to the POSIX operating system API

Portable Operating System Interface (POSIX)

The **readdir**() function returns a pointer to a *dirent* structure representing the next directory entry in the directory stream pointed to by *dirp*. It returns NULL on reaching the end of the directory stream or if an error occurred.

In the glibc implementation, the *dirent* structure is defined as follows:

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* Null-terminated filename */
};
```

Good luck with your midterm :)