String Functions and Tar

Class 25

C-String Library Functions

- there are many useful functions in the string.h library
- strlen: the number of characters before the \0 (remember, useless if the null character is missing)

- strcpy and strncpy: overwrite one string location with the characters of another

- these functions are never safe to just blindly use
- they do no bounds checking and will happily exceed the limits of the array, clobbering any memory in their way
- they have been the source of malicious code of all sorts

- same goes for strcat and strncat

strcmp

- a very useful function
- `int result = strcmp(string1, string2);` returns
  - zero if the two strings are identical up to the null character
  - a negative if string1 is alphabetically before string2
  - a positive is string1 is alphabetically after string2

strcmp

- a very useful function
- `int result = strcmp(string1, string2);` returns
  - zero if the two strings are identical up to the null character
  - a negative if string1 is alphabetically before string2
  - a positive is string1 is alphabetically after string2

- confusing: when the strings are identical, strcmp returns zero, which in C normally indicates false

```
if (strcmp("foo", "foo"))
{
  puts("they do NOT match!");
}
else
{
  puts("they DO match!");
}
```

strcmp

- because of this, the code is normally written like this instead:

```
if (strcmp("foo", "foo") == 0)
{
  puts("they DO match!");
}
else
{
  puts("they do NOT match!");
}
```

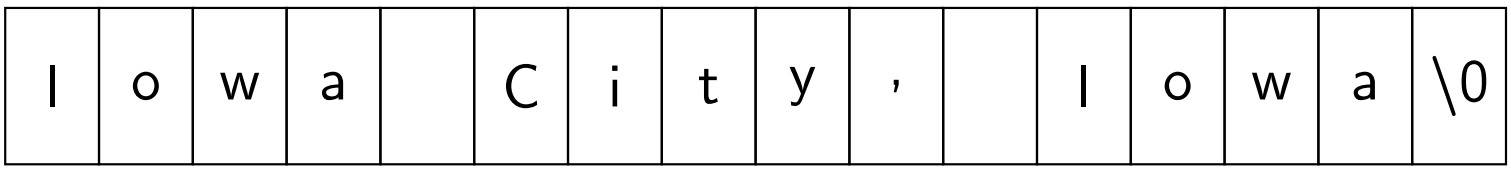
strstr

- strstr: find the location of a substring within a string
- return a pointer to the first character of the substring that matches
- return NULL if the substring is not found

strstr

- let's say you have a string with a city name followed by a comma and space followed by a state name and are trying to find the state name

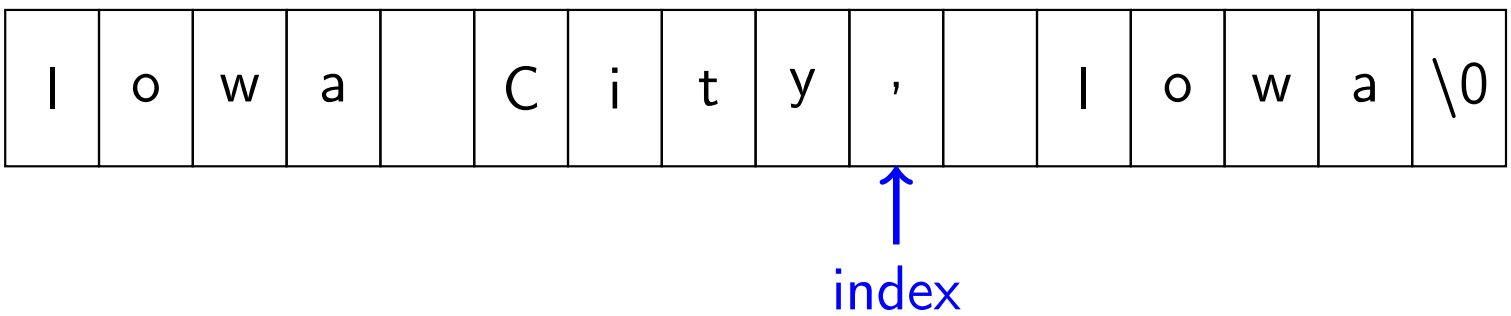
```
char* index = strstr(city_state, ", ");
```



strstr

- let's say you have a string with a city name followed by a comma and space followed by a state name and are trying to find the state name

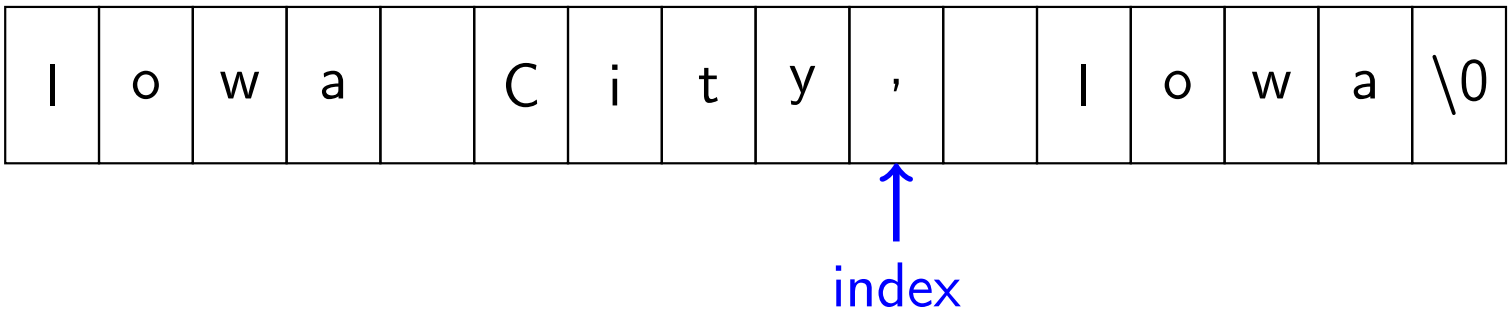
```
char* index = strstr(city_state, ", ");
```



strstr

- let's say you have a string with a city name followed by a comma and space followed by a state name and are trying to find the state name

```
char* index = strstr(city_state, ", ");
printf("%s\n", index[2]);
```



- the output is Iowa
- it's fine to have a pointer pointing into the middle of another string

Parsing

- your current assignment requires you to parse lines that look like this:

```
Harry Potter[TAB]423-749-6268[TAB]boy_who_lived@hp.book
Ginny Weasley[TAB]880-813-1994[TAB]gf@hp.book
Ron Weasley[TAB]756-816-0257[TAB]best_bud@hp.book
Hermione Granger[TAB]243-961-7287[TAB]best_girl@hp.book
```

- must parse lines consisting of fields separated by tabs into separate strings
- you could use strstr for this, and I recommend it
- an alternative is to use strtok
- even though strtok is often used, beware that it is infamous because it breaks its argument string
- this is considered extremely bad design today

```c
#include <stdio.h>
#include <string.h>
int main(void)
{
  /* NOTE the following string has embedded TAB characters */
  char string[] = "This sentence    has    five   tokens.";
  char* token;

  puts(string);
  token = strtok(string, "\t");
  while (token != NULL)
  {
    puts(token);
    token = strtok(NULL, "\t");
  }
  puts(string);
  return 0;
}
```

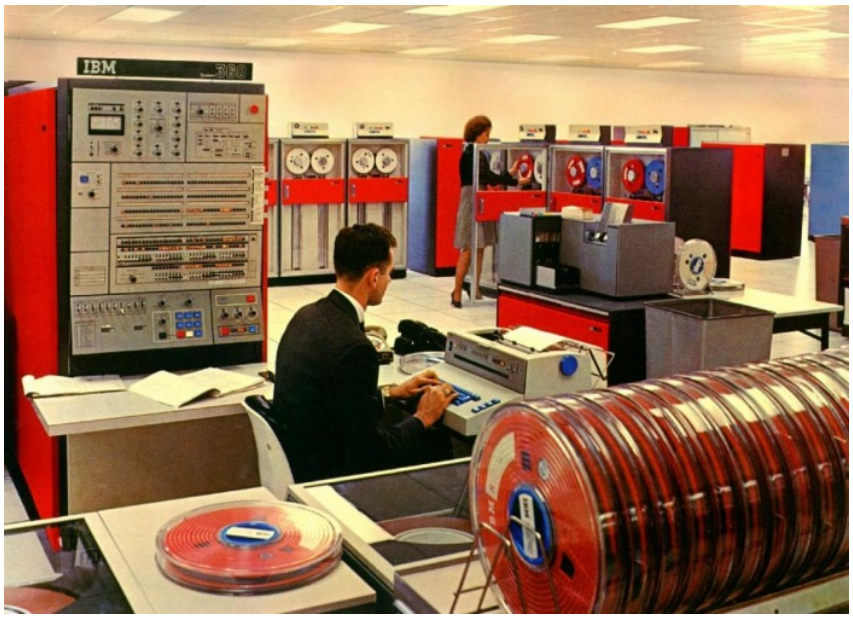
```
This sentence has five tokens.
This
sentence
has
five
tokens.
This
```

Archives

- an archive is a collection of files packaged into a single file
- archives are used for
  - a collection of .o files, called library
  - a backup of a portion of a filesystem
  - distribution of a bunch of files
- you may be familiar with a Java archive, called a jar file
- can be made executable, so that an entire Java system is contained within one file

Archive for Distribution

- on Unix systems, the tar program is universally used for archiving files for distribution
- mnemonic for tape archive, originally written for backup to tape
- tar still has code to control tape machines



tar

- the primary modes tar runs in are
  - c: create a new archive (a tarball)
  - r: append to an archive
  - x: extract files from an archive
  - t: print a table of contents of an archive
- these four modes are mutually exclusive

Create

create an archive name “archive.tar” that contains all files in the current directory, and do it verbosely

```
$ ls
atoh.c  atoh.o  main.o    Makefile.~1~  revbits.c  revbits.o
atoh.h  main.c  Makefile  revbits*       revbits.h

$ tar -cvf archive.tar *.h *.c Makefile
atoh.h
revbits.h
atoh.c
main.c
revbits.c
Makefile

$ ls
archive.tar  atoh.h  main.c  Makefile      revbits*   revbits.h
atoh.c       atoh.o  main.o  Makefile.~1~  revbits.c  revbits.o
```

archive.tar is the argument for the -f option

everything after the last option or argument is a filename

List

list the files that are in an archive

```
$ tar -tf archive.tar
atoh.h
revbits.h
atoh.c
main.c
revbits.c
Makefile
```

list the files verbosely

```
$ tar -tvf archive.tar
-rw-rw-r-- jbeck/jbeck     448 2021-10-22 08:38 atoh.h
-rw-rw-r-- jbeck/jbeck     409 2021-10-22 08:38 revbits.h
-rw-rw-r-- jbeck/jbeck     680 2021-10-22 08:38 atoh.c
-rw-rw-r-- jbeck/jbeck     275 2021-10-22 08:48 main.c
-rw-rw-r-- jbeck/jbeck     469 2021-10-22 08:38 revbits.c
-rw-rw-r-- jbeck/jbeck     435 2021-10-22 08:46 Makefile
```

Extract

cd to a new directory and extract the files from the archive here

```
$ tar -xvf ../my_project/achive.tar
atoh.h
revbits.h
atoh.c
main.c
revbits.c
Makefile
```

by default, tar saves and extracts files with their original timestamp

Clobber

- all well and good, but just like cp or mv, tar will overwrite files of the same name without warning when extracting
- typically this is not what you want
- typically you want tar to first create a subdirectory, and then extract files into that subdirectory
- when creating an archive, tar acts recursively when given a directory name

Don't Clobber

```
$ ls
bar      foo      my_project/

$ ls my_project
archive.tar  atoh.h  main.c  Makefile      revbits*   revbits.h
atoh.c       atoh.o  main.o  Makefile.~1~  revbits.c  revbits.o

$ tar -cvf archive.tar my_project
my_project/
my_project/archive.tar
my_project/Makefile.~1~
my_project/revbits.c
my_project/atoh.h
my_project/atoh.c
my_project/revbits
my_project/revbits.h
my_project/Makefile
my_project/main.c
my_project/atoh.o
my_project/revbits.o
my_project/main.o
```

Cleaning

- this is bad, as now my tarball is full of cruft: .o's, backup files, etc
- so clean the directory first (`$ make clean`)
- before creating a tarball recursively

File Size

```
$ ls -l
-rw-rw-r-- 1 jbeck jbeck 10240 Oct 22 08:53 archive.tar
-rw-rw-r-- 1 jbeck jbeck   680 Oct 22 08:38 atoh.c
-rw-rw-r-- 1 jbeck jbeck   448 Oct 22 08:38 atoh.h
-rw-rw-r-- 1 jbeck jbeck   275 Oct 22 08:48 main.c
-rw-rw-r-- 1 jbeck jbeck   435 Oct 22 08:46 Makefile
-rw-rw-r-- 1 jbeck jbeck   469 Oct 22 08:38 revbits.c
-rw-rw-r-- 1 jbeck jbeck   409 Oct 22 08:38 revbits.h
```

- look at the sizes of a tarball and its constituent files
  tarball: 10,240 bytes
  files: 2,776
- why? the tarball contains metadata on each file
  - timestamp
  - permissions information
  - size and checksum for error detection
- this is necessary, but makes the tarball quite large

Compression

- almost universally, tar files are compressed with gzip
- gzip is a stand-alone compression program, but tar can call it directly with the -z option:

```
$ tar -cvzf archive.tgz *.h *c
$ tar -tvzf archive.tgz
$ tar -xvzf archive.tgz
```

- note that the extension is .tgz

```
$ ls -l
-rw-rw-r-- 1 jbeck jbeck 10240 Oct 22 08:53 archive.tar
-rw-rw-r-- 1 jbeck jbeck 1098 Oct 22 08:53 archive.tgz
```