CSCI-E26: Class 3
Text, Text, Text (arrays, functions, strings)

0. Introduction

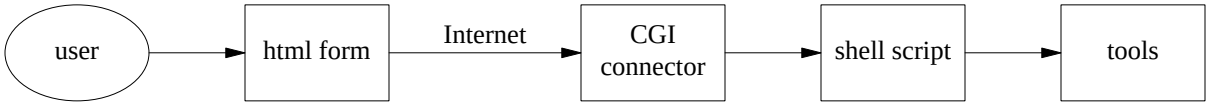
Last week we discussed the idea of programming with software tools, demonstrated some standard Unix tools, and wrote a few software tools using C. This week, we continue using tools to build an interactive website and writing tools in C. We focus on programming with strings.

The outline for tonight's class is:

1. Quick review of the big picture
2. We write a new script -- stationnames
3. HTML and CGI scripts: radio buttons, if then
4. Role of Text, Procssing Text
5. A String is an array of characters terminated by '\0'
6. Arrays, Arrays and Functions
7. Programming with strings
8. What next?

1. Quick Review and CGI without User Input

The big picture for C/Unix/CGI programs is:



We have written two web programs to search the train schedule database: trainsched and train-times.

2. CGI without User Input

We now write a CGI program that does not take any input so does not need a form, stationnames. We build this solution in two parts: a script that reports the list of names and a CGI connector that presents the list in multiple columns.

```
#!/bin/sh
#
# stationnames - list names of stations
#
     cut -d";" -f5 sched | cut -d"=" -f2 | grep '[a-z]' | sort -u
```

The grep command in the pipeline filters out blank lines. Typing stationnames at the command line produces a list of names, one per line. We use this script in a CGI program: stationnames.cgi.

```
#!/bin/sh
# stationnames.cgi
#    outputs list of station names as plain text
#
        echo "Content-type: text/plain"
        echo ""

        ./stationnames | pr -3 -a -t
```

This CGI program does not require user input, so we do not run qryparse. The program prints the content type then produces data by running the stationnames script and sending that output into the tool called pr. The options tell pr to make three columns, to print across then down, and to suppress page titles.

3. Improving Our Website

We now have three CGI programs, one to find times of trains through a station, one to print a schedule, and one to print the names of stations. When we browse to the site, we see a messy list of files. We now begin to improve the site and the forms in three ways:

1. Clearer - Looks better, easier to understand
2. Robust - Less chance for user error
3. Flexible - Give user more control over output

3.1. Clearer - Use Hyperlinks on a page

We write an html page to provide access to these scripts through hyperlinked text:

```
<html>
<head>
     <title>Commuter Rail</title>
     <style type='text/css'>
          A:link { color: darkblue; text-decoration: none; }
          A:visited { color: darkblue;  text-decoration: none; }
          A:hover { text-decoration: underline; color: red; }
     </style>
</head>

<body style='background-color: silver; color: black;
          font-family: sans-serif'>
  <div style='font-weight: bold'>Train Schedule Information</div>

  <div style='margin-left: 30px'>
    <p><a href='train-times.html'>Find train times for a station</a></p>
    <p><a href='trainsched.html'>Find schedule for a train</a></p>
    <p><a href='stationnames.cgi'>List of stations</a></p>
  </div>
</body>
</html>
```

By adding hyperlinks, we present a clear list of functions. The user clicks on a description, not a filename.

3.2. Robust - Use Radio Buttons to Limit Choices

The train-times page asks the user to specify inbound or outbound. The user types the letter 'i' or the letter 'o' into a little text input area. But nothing stops the user from typing in any character. The page would look better, and the user would be less likely to make a mistake if the page had a radio button. Also, the program prints out trains for all days, when the user is more likely to want trains for a specific day. We add two radio buttons increasing clarity, safety, and flexibility:

```
<html>
<body>
Find train times for a station
<p>
<form action="train-times.cgi" method="get">
    Station: <input type="text" size="25" name="station">
    <br>
    Direction: <input type='radio' name='dir' value='i'>Inbound
           <input type='radio' name='dir' value='o'>Outbound
    <br>
    Day:   <input type='radio' name='when' value='m-f'>Weekday
           <input type='radio' name='when' value='sa'>Saturday
           <input type='radio' name='when' value='su'>Sunday
    <br>
    <input type="submit">
</form>
<body></html>
```

We modify the connector CGI script to pass through the 'when' variable:

```
#!/bin/sh
# processing script for train-times.html

    eval `./qryparse`
    echo "Content-type: text/plain"
    echo ""

    echo "Train times for $station direction $dir on $when"
    ./train-times-args "$station" "$dir" "$when"
```

and finally, we modify the actual engine to expect a third argument:

```
#!/bin/sh
#
# train-times-args
#    purpose: list train times for a station
#      usage: train-times-args stationname direction [day]
#      where: direction is "i" or "o", day is "m-f" or "sa" or "su"
#
     # 1st - check there are two arguments or more
     if test $# -lt 2
     then
          echo "usage: train-times-args station direction [day]"
          exit 1
     fi

     STATION=$1
     DIR=$2
     grep "stn=$STATION" sched | grep "dir=$DIR" | grep "day=$3"
```

3.3. Flexible - if..then..else..fi in shell scripts

Next, we make our web page more flexible. The trainsched program uses capitalize to improve the output. Perhaps a user does not want the names capitalized. We give the user the choice by adding a radio button to the form and adding an if..then control structure to the connector script:

```
<html>
<body>
<form action='trainsched2.cgi'>
     Train number please?
     <input type='text' name='trainnumber' size='10'>
     <p>
     Capitalize Names: <input type='radio' name='caps' value='y'>Yes
                 <input type='radio' name='caps' value='n'>No
     <p>
     <input type='submit'>
</form>
</body>
</html>
```

Notice the new variable: caps. This can have the value 'y' or 'n'. We need to check that value in the CGI script so we can invoke capitalize or not:

```
#!/bin/sh
#
# connector for getting train schedule
# this one uses and if then else to capitalize or not
#
     eval `./qryparse`

     echo "Content-type: text/plain"
     echo ""

     if test "$caps" = "y"
     then
          ./trainsched2 $trainnumber | ./capitalize | expand -8
     else
          ./trainsched2 $trainnumber | expand -8
     fi
```

Notice how if..then works in a shell script. In this case, we execute one pipeline if the user selects 'y' and a

different pipeline if the user selects 'n'. Any number of commands can be executed in the then part or in the else part. In fact, any number of commands may be included in the if part, but people rarely do that.

4. Role of Text, Processing Text

Notice the role of text: The datafile is text with a simple, regular format. The HTML form uses text to define what the user sees, the name of the user input fields, and what program to run when the request is sent. Finally, the scripts that process the request are text files that contain Unix commands. The web runs on plain text. But how does that actually work? How do computers store and process strings of characters?

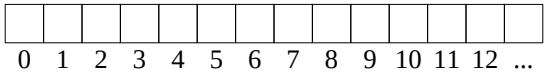
5. How Does C Store Strings of Text?

C has a small number of data types and data structures. The basic data types are integer, floating point number, character, and pointer. Each of these types is supported directly by the hardware. The only two aggregate types are the array and the struct. In C, there are no abstract types with hidden internal structure. In C, a string is just an array of characters with a terminating zero-value byte. All operations on strings are done by system-provided or user-written functions.

Since strings are arrays, we shall discuss the more general case first: arrays and functions. Then we shall return to strings as a special case of arrays and functions.

6. Array and Arrays and Functions: Arrays in C

Arrays in C are easy to understand. All you really need to know is that computer memory looks like this:



Computer memory is a sequence of numbered storage boxes, each holds one byte. On a machine with 2 gigabytes of memory, there are about 2 billion of these boxes numbered 0,1,2, ..., 2billion. An array is just a tiny section of this enormous array. The array starts at some spot in memory and consists of some number of memory boxes. That's the view at the computer hardware level.

The view from the C programming level is that an array is an collection of variables of one type. Each element in the collection is identified by an index. The index begins at 0. Each element may use more than one byte. For example, on a 32-bit machine, an integer usually is stored in 4 bytes.

Creating Create an array with this this syntax:

```
char  x[10];              /* an array of 10 chars          */
int   y[200];             /* an array of 200 ints          */
float z[1];               /* an array of 1 float           */
```

You can create multidimensional arrays with this syntax:

```
char  days[7][12];         /* an array of 7 items, each a 12-char array */
int   temps[12][31];       /* an array of 12 items, each a 31-int array */
char  food[12][31][10];    /* an array of 12 items, each an array of 31 ten-char arrays */
```

Indexing Refer to items in an array by using the square brackets:

```
x[3] = 'A';               /* set fourth element to 'A'    */
s = y[2] + y[12];         /* add two elements from y      */
z[0] = 3.14159;           /* store a float in z[0]        */
strcpy(days[0],"mon");    /* copy a string into an array  */
temps[9][7] = 55;         /* store a temp in a 2d array   */
strcpy(food[3][4],"tuna"); /* copy a string into an array item */
```

Note: the index does not have to be within the length of the array,

```
x[200] = 'B';
y[-3] = 22;
```

but these produce unpredictable results.

Initialize Arrays may be initialized when defined. Char arrays may be initialized with string notation:

```
int   x[4] = { 3, 8, 2, 5 };  /* set all four items    */
char  z[10] = "cookie";       /* only 7 spots are used */
```

Arrays and Functions

You may pass arrays to functions. Arrays are the only C data type that is passed by reference. All other data types are passed by value. Therefore, in this program:

```
#include <stdio.h>

/* array-func-demo.c
 * shows that an int is passed by value, but an array is passed by reference
 * That is, the function can modify the array.
 */
main()
{
     int  n = 4;
     int  a[10] = { 1, 2, 3, 4, 5 };

     printf("n is %d, a is %d, %d, %d, ... \n", n, a[0],a[1],a[2]);
     func(n,a);
     printf("n is %d, a is %d, %d, %d, ... \n", n, a[0],a[1],a[2]);
}
func(int x, int l[10])
{
     printf("x is %d, l is %d, %d, %d, ... \n", x, l[0],l[1],l[2]);
     x = 100;
     l[0] = 36; l[1] = 49; l[2] = 64;
     printf("x is %d, l is %d, %d, %d, ... \n", x, l[0],l[1],l[2]);
}
```

That is everything about arrays. They are simple. They come in any number of dimensions, they are passed by reference, and the C compiler never checks for bounds problems.

7. A String is an Array of Characters Terminated by '\0'

We continue writing programs that work with strings. In particular we shall write programs to analyze/clean data and programs to reformat data. First, we focus on the technical details of strings in C.

Fact One: A string in C is an array of chars. The end of the string is marked by the character with code 0, written as '\0'.

Fact Two: Operations on strings are simple array operations. To help programmers, All C systems include a standard string library. This library includes several functions for doing things with strings. These functions include:

Some String Library Functions

strlen(s)	returns length of s
strcpy(des,src)	copies string src into array des
strcmp(s1,s2)	compares strings s1 and s2
strcat(des,src)	appends string src to end of des
fgets(s,len,fp)	reads a line of text into string s

Fact Three: Strings may not be copied with =. You must use strcpy() to copy an array of chars.

8. Programming with Strings

The sched datafile may have fields in the wrong order. We depend on a fixed order to select fields using the cut command. If the station name were in column 3 rather than column 5, we'd have trouble.

String Input and Output

Everything about arrays applies to strings. Strings have special functions for input and output, though.

Input	Use fgets(s,len,stdin) to read a line of text from standard intput into an array s of length len. fgets stops at a newline or at len-1 chars. fgets puts a nul byte in the array.
Output	You can use printf with %s to print a string, or you can use fputs(s,stdout).

Writing a String Parsing Function

We shall now write a C function to split a line of delimited text into separate strings. The function is called splitline(). It takes a single line of text and copies the separate items into an array of char arrays.

Using a String Parsing Function - data cleaning

We can write a program to use this function in a different C file and compile the two files together:

```
cc fix_sched_order.c splitline.c -o fix_sched_order
```

Writing fgets

fgets is a function written in C and stored in the C library. When you compile your code, the compiler brings in (links) the code for fgets from the external file it lives in. But, you could write your own line reading function. We write readln.c and use it to find empty fields.

9. What Next?

We made our website cleaner and clearer. We added radio buttons to make our page safer and more flexible. We saw how to process strings and arrays with C functions, and we saw how to build utility functions in their own files. Our next task is to make the output of our schedule searches look better. We shall replace the plain text responses with nice-looking html tables. For the next assignment, you will write a tool to do that.