COMP1917: Computing 1

10. Strings and Files

Reading: Moffat, Section 7.6-7.10, Chapter 11.

Outline

- String format
- Character pointers
- String functions
- Command-line arguments (using `argc` and `argv[]`)
- Scanning strings (using `gets()`, `fgets()`, `scanf()`)
- `FILE` input and output

Strings

- “string” is a special word for an array of characters
- when a string is printed or copied, we need to know when to stop
- rather than store the length of the string separately, we instead use a special “end-of-string” marker similar to end-of-file
- end-of-string is denoted by `'\0'` and is always implemented as 0

String Example

Suppose we have a character array `s`

```
char s[12];
```

If `s[]` contained the string “hello”, this is how it would look in memory:

0	1	2	3	4	5	6	7	8	9	10	11
h	e	l	l	o	\0						

String Constants

Note the difference between a character constant like `'x'` and a string constant like `"x"`.

Note that the following declaration is fine

```
char str[] = "hello world";
```

C will determine how much storage space to allocate.

Character Pointers

Strings can be scanned and printed with `scanf()` and `printf()`, using the format `"%s"`

```
char *p = "Cheshire:-)";
while( *p != '\0' ) {
   printf( "p = %X, string at p = %s\n", p, p );
   p++;
}
```

Character Pointers

```
p = 1FD8, string at p = Cheshire:-)
p = 1FD9, string at p = heshire:-)
p = 1FDA, string at p = eshire:-)
p = 1FDB, string at p = shire:-)
p = 1FDC, string at p = hire:-)
p = 1FDD, string at p = ire:-)
p = 1FDE, string at p = re:-)
p = 1FDF, string at p = e:-)
p = 1FE0, string at p = :-)
p = 1FE1, string at p = -)
p = 1FE2, string at p = )
```

String Functions

C provides a number of string manipulation functions:

```
strlen()  // length of string
strcpy()  // copy one string to another
strcat()  // concatenate two strings
strchr()  // find character in string
strcmp()  // compare two strings
strstr()  // find substring inside string
```

It is instructive to write our own (perhaps simplified) versions of these functions.

Length of a String — `strlen()`

```
#include <string.h>
int strlen( const char s[] );
```

Returns number of characters in string `s` up to but not including `'\0'`

```
int strlen( const char s[] )
{
    int len;

    for( len = 0; s[len] != '\0'; len++ )
        ;
    return( len );
}
```

Array and String Implementations

Many string manipulation functions can be implemented using either arrays of `char` or pointers to `char`.

The pointer versions are often shorter and in some ways more “elegant” (but may expose us to the danger of computer viruses and buffer-overrun attacks).

Copying a String — `strcpy()`

```
#include <string.h>
char *strcpy(char dest[], const char src[]);
```

- copies string contained in `src` into string (i.e. array) `dest` including the terminating `'\0'`.
- returns start of string `dest`.
- you must ensure that `dest` is large enough to accept all of `src`.

Two (simplified) versions of `strcpy()`

```
strcpy( char s[], char t[] ) // array version
{
   int i = 0;
   // increment the array index as you copy
   while(( s[i] = t[i] ) != '\0' )
      i++;
}
strcpy( char *s, char *t ) // pointer version
{  // increment the pointers as you copy
   while(( *s = *t ) != '\0' ) {
      s++;
      t++;
   }
}
```

Concatenating Strings — `strcat()`

```
#include <string.h>
char *strcat(char dest[], const char src[]);
```

- appends string `src` to the end of `dest` overwriting the `'\0'` at the end of `dest` and adds terminating `'\0'`.
- returns start of string `dest`.
- you must ensure `dest` is large enough to add `src` to the end of it.

Concatenating Strings — `strcat()`

Example

`dest =`

0	1	2	3	4	5	6	7	8	9	10	11	12	13
p	o	w	d	e	r	\0							

`src =`

0	1	2	3	4	5	6	7	8	9	10
f	i	n	g	e	r	\0				

`strcat(dest,src)`

0	1	2	3	4	5	6	7	8	9	10	11	12	13
p	o	w	d	e	r	f	i	n	g	e	r	\0	

Pointer Version of `strcat()`

Function to append one string to another

```
strcat( char *s, char *t )  // pointer version
{
   while( *s != '\0' ) {    // scan to the end of s
      s++;
   }
   // make use of the fact that '\0' is actually zero
   while( *s = *t ) // now copy t to end of s
      s++;
      t++;
   }
}
```

Pointer Version of `strchr()`

Function `strchr()` – return a pointer to the first occurrence of the character `ch` in the string `s`, or `NULL` if `ch` does not occur in `s`.

```
char * strchr( char *s, int ch )
{
   while( *s != '\0' && *s != ch ) { // scan until you find ch
      s++;
   }
   if( *s == ch ) {
      return( s );  // return s if ch is found
   }
   else {
      return NULL;  // otherwise, return NULL
   }
}
```

Comparing Strings — `strcmp()`

```
#include <string.h>
int strcmp(const char s1[], const char s2[]);
```

- compares strings `s1` and `s2` until `'\0'` is encountered in both (in which case they are identical) or until a difference is found.
- returns a value based on this comparison:

Comparison	Result Returned
`s1 > s2`	integer greater than 0
`s1 < s2`	integer less than 0
`s1 = s2`	0

- integer returned is the difference of the characters on which `s1` and `s2` differ or 0 if they are identical.

Pointer version of `strcmp()`

```
int strcmp( char *s, char *t )
{
   // find the first character where s and t differ
   while( *s != '\0' && *s == *t ) {
      s++;
      t++;
   }
   // return difference between the two characters
   return( *s - *t );
}
```

Safe copying — `strncpy()` and `strncat()`

`strcpy()` and `strcat()` are not safe because they can easily run over the end of the array, thus exposing your computer to a "buffer overrun attack".

You should instead use `strncpy()` and `strncat()`, which put a limit on the number of characters copied.

`man strncpy`:

> The `strncpy()` function copies at most `n` characters from `src` into `dest`. If `src` is less than `n` characters long, the remainder of `dest` is filled with `'\0'` characters. Otherwise, `dest` is not null-terminated.

Safely copying a String — `strncpy()`

```
char *strncpy( char dest[], const char src[], int n )
{
  int i;
  // check index is not too large, before copying
  for( i=0; ( i < n )&&( src[i] != '\0' ); i++ ) {
     dest[i] = src[i];
  }
  // pad any remaining space with null characters
  for( ; i < n; i++ ) {
     dest[i] = '\0';
  }
  return( dest );
}
```

Using `argc` and `argv`

- the parameters `argc` and `argv` convey the command line arguments used when a program is executed:

```
int main( int argc, char *argv[] )
{ ... }
```

- the integer `argc` indicates the number of parameters found on the command line.
- the array `argv[]` stores each command line argument as a string.
- Note that `argv[0]` is the name of the executable; the actual arguments are `argv[1]`, `argv[2]`, etc.
- in other words, `argc` counts one more argument than you might expect; if there are no arguments, `argc` will be 1.

Program to echo its arguments

```
int main( int argc, char *argv[] )
{
   int i;

   for( i = 0; i < argc; i++ ) {
      printf( "arg %d: %s\n", i, argv[i] );
   }
}
```

```
$ ./a.out abc 123 Do_re_mi
arg 0: ./a.out
arg 1: abc
arg 2: 123
arg 3: Do_re_mi
```

`gets()` and `fgets()`

```
#include <stdio.h>
char *gets(char *s)
```

Reads a line from standard input into the character array `s` until newline or end of file is encountered.

`man gets`:

> Never use gets(). Because it is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use fgets() instead.

Example of `fgets()`

Here is a simple example of `fgets()`:

```
int main(void)
{
    char s[128];   // string array

    // get max of 128 chars from Standard Input
    fgets( s, 128, stdin );
    printf( "%s", s );     // newline is part of string

    return  0;
}
```

Scanning strings with `scanf()`

If you want to scan just a single word rather than a whole line, you can use `scanf("%s")`

```
int main(void)
{
    int num;
    char s[21];    // string array

    // read a number, then a word from stanard input
    scanf("%d %20s", &num, s); // limit to 20 chars
    printf("Number was %d, word was %s\n", num, s);

    return 0;
}
```

More about `scanf()`

- `scanf()` doesn't need to know the previous value of the variable; it needs to know the address for storing the new value

```
scanf("%d", &num );
```

- it is also fine to use a pointer

```
int *p = &num;
scanf("%d", p );
```

- when we scan a string, we don't need the ampersand because the string is already a pointer

```
scanf("%s", s );
```

What is a File ?

- a sequential stream of bytes
- used for permanent retention of data
- identified by a file name and, in a program, by a file pointer
- may be classified as
  - either text or binary
  - either sequential access or random access
- C has three standard text files:
  - `stdin` (standard input)
  - `stdout` (standard output)
  - `stderr` (standard error)

Files

- we can often achieve what we want by “redirecting” standard input and output using < and >
- however, there are some situations where we need to read or write to a file explicitly
- files other than `stdin`, `stdout` and `stderr` can be opened with `fopen()` and closed with `fclose()`
- we can scan from and print to files using `fscanf()` and `fprintf()`
- we can read or write a single character using `getc()` and `putc()`
- we can read or write a string using `fgets()` and `fputs()`

Program to copy one file to another

```
int main( int argc, char *argv[] )
{
   FILE *infile, *outfile;
   int ch;

   if( argc < 3 ) { // check there are enough arguments
      printf("Usage: %s <infile> <outfile>\n", argv[0] );
   }
   infile = fopen( argv[1], "r" ); // open for reading
   if( infile == NULL ) {
      printf("Error: file not found: %s\n", argv[1] );
      exit( 1 );
   }
```

Program to copy one file to another

```
    outfile = fopen( argv[2], "w" ); // open for writing
    if( outfile == NULL ) {
        printf( "Error opening file %s\n", argv[2] );
        exit( 1 );
    }
    while(( ch = getc( infile )) != EOF ) {
        putc(ch,outfile); // copy one character at a time
    }
    fclose( infile );
    fclose( outfile );   // close files when you finish
}
```