Week 9 Lecture 2

ASCII Files

Files in C

Command I/O

- Remember
  - The standard input is stdin
  - The standard output is stdout
  - Standard error is called stderr
- These are the default files you can read from and write to
- You can redirect the input using < and >

Using files in the program

- You can also name files in a program.
- To use a named file.
  - Create a file pointer using fopen
  - Read characters from the file pointer
  - Close the file pointer when finished

Creating a file pointer with fopen

- Template:
  - FILE *fopen(char *filename, char *mode);
- The function fopen
  - Returns a file pointer (i.e., File *)
  - Takes a file path (e.g. /temp/inputfile)
  - Takes a mode: (r, w, a, r+, w+, a+)

Mode

- Mode determines what you can do with the file
  - Read: r
  - Write: w (Creates new file if non exists)
  - Append: a (Creates new file if none exists)
  - Read and Write: r+
  - Read and Write: w+ (Zeros the file if one exists creates if it does not.)
  - Read and append: a+ (Creates new file if none exits; reads from beginning but appends to end.)

File pointer

- Store the return value as a FILE *
- The variable is the argument to function that read, write and append.
- It is also the argument to close().

Reading Files

Reading

- Functions that read from a FILE *
  - fgetc(FILE *fp)
    - Returns character
  - fgets(FILE *fp)
    - Returns string
  - fscanf(FILE *fp, “<ctrl>”, var…)
    - Returns number of characters read

Example: print file to monitor

- FILE * var
- Open the file.
  - Mode = “r”
- Read the characters.
- Close the file.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *fp;
  char ch;

  fp = fopen("temp", "r");

  while ((ch = getc(fp)) != EOF) {
    printf("%c", ch);
  }

  fclose(fp);

  return 0;
}
```

Error checking

- File operations are I/O operation so unexpected things can happen.
- Always check for errors
  - The fopen function returns NULL when it fails
  - The fclose function returns EOF when it fails

Improved example

- Check that the file was opened.
  - Stop the program it isn't
- Check that the file was closed.
  - Stop the program if it isn't.
- Return 0 if program succeeds

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *fp;
  char ch;

  if ((fp = fopen("temp", "r")) == NULL) {
    fprintf(stderr, "failed to open temp");
    return 1;
  }

  while ((ch = getc(fp)) != EOF) {
    printf("%c", ch);
  }

  if (fclose(fp) == EOF) {
    printf("Failed to close fp");
    return 1;
  }

  return 0;
}
```

Writing Files

Writing

- Functions that write to a FILE *
  - fputc(char ch, FILE *fp)
  - fputs(char *s, FILE *fp)
  - fprintf(FILE *fp, “<ctrl>”, var…)
    - Returns number of chars printed
- The character to read or write is passed in as a parameter.
- The put and puts functions return EOF on error.

Example: type into file

- Create a FILE *
- Open the file
  - Checking that it is open
- Put the char from the terminal in the file
  - Checking that we got a character
- Close the file.
  - Checking that it is really closed

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *fp;
  char ch;

  if ((fp = fopen("temp", "w")) == NULL) {
    fprintf(stderr, "failed to open temp");
    return 1;
  }

  printf("Enter data: ");
  while ((ch = getchar()) != EOF) {
    putc(ch, fp);
  }

  if (fclose(fp) == EOF) {
    printf("Failed to close fp1");
    return 1;
  }

  return 0;
}
```

Reading and Writing Strings

Strings

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *fp;
  char *s;
  int i;

  if ((fp = fopen("temp", "w")) == NULL) {
    fprintf(stderr, "failed to open temp");
    return 1;
  }

  s = "Hello world\n";
  for (i = 0; i < 10; i++) {
    fputs(s, fp);
  }

  if (fclose(fp) == EOF) {
    printf("Failed to close fp1");
    return 1;
  }

  return 0;
}
```

```
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
temp (END)
```

Why use ASCII Files?

- Characters are primitive types that can represented almost anything.
  - Text files can be written by one program and read by another that knows nothing of the first
  - Unix provides pipes that send the standard output of one file into the standard input of the next
  - Using pipes you can perform complex processing using small simple programs.