File Handling

IC-100

January. 2023

Today

- File Processing
- Multi File Programming
- Makefile Utility

Files

- What is a file?
  - Collection of bytes stored on secondary storage like hard disks (not RAM).
- Any *addressable* part of the file system in an Operating system can be a file.
  - includes such strange things as /dev/null (nothing), /dev/usb (USB port), /dev/audio (speakers)

File Access

- 3 files are always connected to a C program :
  - stdin : the standard input, from where scanf, getchar(), gets() etc. read input from
  - stdout : the standard output, to where printf(), putchar(), puts() etc. output to.
  - stderr : standard error console.

File Handling in C

1. Open the file for reading/writing etc.: fopen
   - return a *file pointer*
   - pointer points to an internal structure containing information about the file:
     - location of a file
     - the current position being read in the file, etc.

   FILE* fopen (char *name, char *mode)

2. Read/Write to the file

   int fscanf(FILE *fp, char *format, …)

   int fprintf(FILE *fp, char *format, …)

   int fputs(const char* str, FILE *fp)

3. Close the File.

   int fclose(FILE *fp)

Compared to scanf and printf – a new (first) argument fp is added

Opening Files

FILE* fopen (char *name, char *mode)

- The first argument is the name of the file
  - can be given in short form (e.g. “inputfile”) or the full path name (e.g. “/home/don/inputfile”)
- The second argument is the mode in which we want to open the file. Common modes include:
  - “r” : read-only. Any write to the file will fail. File must exist.
  - “w” : write. The first write happens at the beginning of the file, by default. Thus, may overwrite the current content. A new file is created if it does not exist.
  - “a” : append. The first write is to the end of the current content. File is created if it does not exist.

Opening Files

- If successful, fopen returns a *file pointer* – this is later used for fprintf, fscanf etc.
- If unsuccessful, fopen returns a NULL.
- It is a good idea to check for errors (e.g. Opening a file on a CDROM using "w" mode etc.)

Closing Files

- An open file must be closed after last use
  - allows reuse of FILE* resources
  - flushing of ***buffered*** data (to actually write!)

File I/O: Example

- Write a program that will take two filenames, and print contents to the standard output. The contents of the first file should be printed first, and then the contents of the second.
- The algorithm:
  1. Read the file names.
  2. Open file 1. If open failed, we exit
  3. Print the contents of file 1 to stdout
  4. Close file 1
  5. Open file 2. If open failed, we exit
  6. Print the contents of file 2 to stdout
  7. Close file 2

```
int main()
{
  FILE *fp;  char filename1[128], filename2[128];
  scanf("%s", filename1);
              scanf("%s", filename2);
              fp = fopen( filename1, "r" );
  if(fp == NULL) {
            fprintf(stderr, "Opening File %s failed\n", filename1);
            return -1;
  }
  copy_file(fp, stdout);
  fclose(fp);
              fp = fopen( filename2, "r" );
  if (fp == NULL) {
            fprintf(stderr, "Opening File %s failed\n", filename2);
                        return -1;
  }
  copy_file (fp, stdout);
  fclose(fp);
              return 0;
}
```

The Program: copy_file

```
void copy_file(FILE *fromfp, FILE *tofp)
{
  char ch;

  while ( !feof ( fromfp ) ) {
    fscanf  ( fromfp, "%c", &ch );
    fprintf ( tofp, "%c", ch );
  }
}
```

Some Other File Handling Functions

- `int feof ( FILE* fp );`
  - Checks whether the EOF is set for fp – that is, the EOF has been encountered. If EOF is set, it returns nonzero. Otherwise, returns 0.
- `int ferror ( FILE *fp );`
  - Checks whether the error indicator has been set for fp. (for example, write errors to the file.)

Some Other File Handling Functions

- **`int fseek(FILE *fp, long int offset, int origin);`**
  - To set the current position associated with fp, to a new position = origin + offset.
  - Origin can be:
    - SEEK_SET: beginning of file
    - SEEK_CURR: current position of file pointer
    - SEEK_END: End of file
  - Offset is the number of bytes.
- **`int ftell(FILE *fp)`**
  - Returns the current value of the position indicator of the stream.

Opening Files: More Modes

- There are other modes for opening files, as well.
  - "r+" : open a file for read and update. The file must be present.
  - "w+" : write/read. Create an empty file or overwrite an existing one.
  - "a+" : append/read. File is created if it doesn't exist. The file position for reading is at the beginning, but output is appended to the end.

File I/O Example

```
#include <stdio.h>
int main () {
        FILE * fp = fopen("file.txt","w+");
        fputs("This is Hebrew Language", fp);
        fseek( fp, 7, SEEK_SET );
        fputs(" C Programming Language", fp);
        fclose(fp);

        int c;
        fp = fopen("file.txt","r");
        while(1)  {
                c = fgetc(fp);
                if( feof(fp) ) break;
                printf("%c", c);
        }
        fclose(fp);
        return 0;
}
```

This is C Programming Language

File I/O: stdout vs stderr

- What is the output of following program when run on a terminal:

```
#include <stdio.h>
int main()
{
  int input;
  scanf("%d", &input);
  fprintf(stdout, "Printing to STDOUT %d\n", input);
  fprintf(stderr, "Printing to STDERR %d\n", input);
  return 0;
}
```

INPUT
5

Printing to STDOUT 5
Printing to STDERR 5

File I/O: stdout vs stderr

- What is the output of following program when run on a terminal:

```
#include <stdio.h>
int main()
{
  int input;
  scanf("%d", &input);
  fprintf(stdout, "Printing to STDOUT %d", input);
  fprintf(stderr, "Printing to STDERR %d", input);
  return 0;
}
```

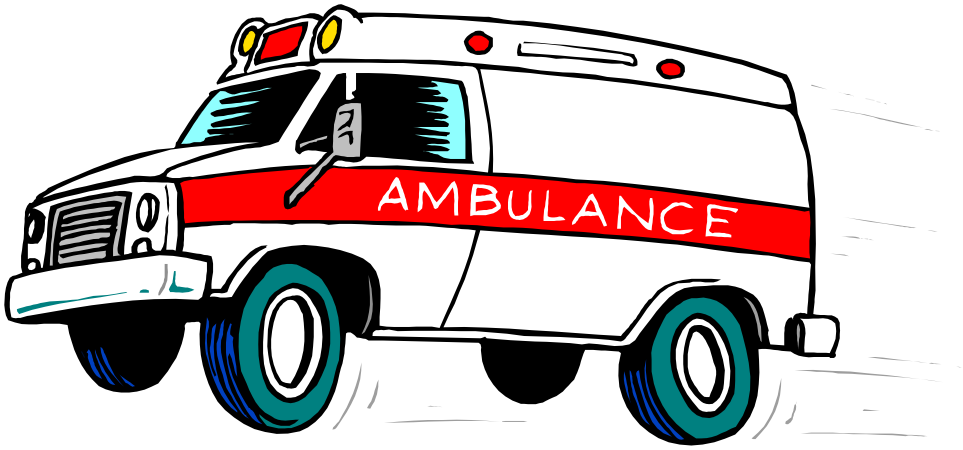
INPUT
5

~~Printing to STDOUT 5Printing to STDERR 5~~

Printing to STDERR 5Printing to STDOUT 5

Stdout vs. Stderr (Intuition)

vs.



An Exercise

- Write a C program to create class student database

1. Create student.doc file.
2. Add student data 1 by 1
3. Read from student.doc and display to user

Multi-File Program

```
#include <stdio.h>
void hello () {
  printf("Hello World!!\n");
}
int factorial(int n){
    if(n!=1){
        return(n * factorial(n-1));
    }
    else return 1;
}
int main(){
    hello();
    printf("The factorial of 5 is %d\n", factorial(5));
    return 0;
}
```

Multi-File Program

```
#include<stdio.h>

void hello () {
  printf("Hello World!!\n");
}
```

Hello.c

```
int factorial(int n){
    if(n!=1){
        return(n * factorial(n-1));
    }
    else return 1;
}
```

fact.c

Multi-File Program

```
#include <stdio.h>
#include "fact.h"
#include "hello.h"

int main(){
    hello();
    printf("The factorial of 5 is %d\n", factorial(5));
    return 0;
}
```

main.c

```
void hello ();
```

Hello.h

```
int factorial(int );
```

fact.h

How to Compile?

hello.c
hello.h

fact.c
fact.h

main.c

```
gcc –o out  hello.c fact.c main.c

gcc –c hello.c    // Produces hello.o
gcc –c fact.c
gcc –c main.c
gcc –o out hello.o fact.o main.o
```

Makefile

- Automation of build process
- Used to compile large projects with many source files

Makefile Example

```
all: hello

hello: main.o fact.o hello.o
	gcc main.o fact.o hello.o -o hello

main.o: main.c fact.h hello.h
	gcc -c main.c

fact.o: fact.c
	gcc -c fact.c

hello.o: hello.c
	gcc -c hello.c

clean:
	rm *o hello
```

Makefile Example

```
CC=gcc
CFLAGS=-c -Wall

all: hello

hello: main.o fact.o hello.o
	$(CC) main.o fact.o hello.o -o hello

main.o: main.c fact.h hello.h
	$(CC) $(CFLAGS) main.c

fact.o: fact.c
	$(CC) $(CFLAGS) fact.c

hello.o: hello.c
	$(CC) $(CFLAGS) hello.c

clean:
	rm *o hello
```

Make command

$ make

$ make –f myMakefile

$ make clean

Use of Make command

```
$ make
gcc -c main.c
gcc -c fact.c
gcc -c hello.c
gcc main.o fact.o hello.o -o hello
$

$ vi hello.c        // update hello.c
$ make
gcc -c hello.c
gcc main.o fact.o hello.o -o hello
$
```

Use of Make command

```
$$ make clean
rm *o hello
$
```

```
$ make
gcc -c main.c
gcc -c fact.c
gcc -c hello.c
gcc main.o fact.o hello.o -o hello
$
```

Make Utility: Further Reading

- https://www.gnu.org/software/make/manual/make.html