
C Basic File Input/Output Manipulation

C Programming - File Outline

- ❖ File handling in C - opening and closing.
- ❖ Reading from and writing to files.
- ❖ Special file streams stdin, stdout & stderr.
- ❖ How we *SHOULD* read input from the user.
- ❖ What are STRUCTURES?
- ❖ What is dynamic memory allocation?

File handling in C

- ❖ In C we use `FILE *` to represent a pointer to a file.
- ❖ `fopen` is used to open a file. It returns the special value `NULL` to indicate that it couldn't open the file.

```
FILE *fptr;  
char filename[] = "file2.dat";  
fptr = fopen(filename, "w");  
if (fptr == NULL) {  
    fprintf(stderr, "ERROR");  
    /* DO SOMETHING */  
}
```

Modes for opening files

- ❖ The second argument of `fopen` is the *mode* in which we open the file. There are three
- ❖ "r" opens a file for reading
- ❖ "w" opens a file for writing - and writes over all previous contents (deletes the file so be careful!)
- ❖ "a" opens a file for appending - writing on the end of the file

The exit() function

- ❖ Sometimes error checking means we want an "emergency exit" from a program. We want it to stop dead.
- ❖ In main we can use "return" to stop.
- ❖ In functions we can use exit to do this.
- ❖ Exit is part of the `stdlib.h` library

`exit(-1);`

in a function is exactly the same as

`return -1;`

in the main routine

Writing to a file using fprintf

- ❖ fprintf works just like printf and sprintf except that its first argument is a file pointer.
- ❖ We could also read numbers from a file using fscanf - but there is a better way.

```
FILE *fptr;  
fptr= fopen ("file.dat","w");  
/* Check it's open */  
fprintf (fptr,"Hello World!\n");
```

Reading from a file using fgets

- ❖ fgets is a better way to read from a file
- ❖ We can read into a string using fgets

```
FILE *fptr;  
char line [1000];  
/* Open file and check it is open */  
while (fgets(line,1000,fptr) != NULL) {  
    printf ("Read line %s\n",line);  
}
```

fgets takes 3 arguments, a string, a maximum number of characters to read and a file pointer. It returns NULL if there is an error (such as EOF)

Closing a file

- ❖ We can close a file simply using `fclose` and the file pointer. Here's a complete "hello files".

```
FILE *fptr;  
char filename[] = "myfile.dat";  
fptr = fopen(filename, "w");  
if (fptr == NULL) {  
    printf("Cannot open file to write!\n");  
    exit(-1);  
}  
fprintf(fptr, "Hello World of filing!\n");  
fclose(fptr);
```


Open and Close

- ❖ We use the file pointer to close the file - not the name of the file

```
FILE *fptr;  
fptr= fopen ("myfile.dat","r");  
/* Read from file */  
fclose (fptr);
```

Three special streams

- ❖ Three special file streams are defined in the `stdio.h` header
- ❖ `stdin` reads input from the keyboard
- ❖ `stdout` send output to the screen
- ❖ `stderr` prints errors to an error device (usually also the screen)
- ❖ What might this do:

```
fprintf(stdout, "Hello World!\n");
```

Reading loops

- ❖ It is quite common to want to read every line in a program. The best way to do this is a while loop using fgets.

```
/* define MAXLEN at start using enum */
FILE *fptr;
char tline[MAXLEN]; /* A line of text */
fptr= fopen ("sillyfile.txt","r");
/* check it's open */
while (fgets (tline, MAXLEN, fptr) != NULL) {
    printf ("%s",tline); // Print it
}
fclose (fptr);
```

Using fgets to read from the keyboard

- ❖ fgets and stdin can be combined to get a safe way to get a line of input from the user

```
#include <stdio.h>
int main()
{
    const int MAXLEN=1000;
    char readline[MAXLEN];
    fgets (readline,MAXLEN,stdin);
    printf ("You typed %s",readline);
    return 0;
}
```

Getting numbers from strings

- ❖ Once we've got a string with a number in it (either from a file or from the user typing) we can use `atoi` or `atof` to convert it to a number
- ❖ The functions are part of `stdlib.h`

```
char numberstring[] = "3.14";  
int i;  
double pi;  
pi = atof(numberstring);  
i = atoi("12");
```

Both of these functions return 0 if they have a problem

Issues

- ❖ fgets includes the '\n' on the end
- ❖ This can be a problem - for example if in the last example we got input from the user and tried to use it to write a file:

```
FILE *fptr;  
char readfname[1000];  
fgets (readfname,1000,stdin);  
fptr= fopen (readfname,"w");  
/* oopsie - file name also has \n */
```

Even experienced programmers can make this error

Dynamic memory allocation

- ❖ How would we code the "sieve of Eratosthenes" to print all primes between 1 and n where the user chooses n ?
- ❖ We could simply define a HUGE array of chars for the sieve - but this is wasteful and how big should HUGE be?
- ❖ The key is to have a variable size array.
- ❖ In C we do this with DYNAMIC MEMORY ALLOCATION

A dynamically allocated sieve

```
#include <stdlib.h>
void vary_sieve (int);
void vary_sieve (int n)
/* Sieve to find all primes from 1 - n */
{
    char *sieve;
    int i;
    sieve= (char *)malloc (n*sizeof(char));
    /* check memory here */
    for (i= 0; i < n; i++)
        sieve[i]= UNCROSSED;
    /* Rest of sieve code */
    free (sieve); /* free memory */
}
```


A closer look at malloc

- ❖ Let's look at that malloc statement again:

sieve= (char *)malloc(n*sizeof(char));

↖
This is a CAST
(remember them)
that forces the variable
to the right type (not
needed)

↖
we want
n chars

↖
sizeof(char) returns how
much memory a char
takes

This says in effect "grab me enough memory for 'n' chars"

Free

- ❖ The free statement says to the computer "you may have the memory back again"

```
free(sieve);
```

essentially, this tells the machine that the memory we grabbed for sieve is no longer needed and can be used for other things again. It is `_VITAL_` to remember to free every bit of memory you malloc

Check the memory from malloc

- ❖ Like fopen, malloc returns NULL if it has a problem
- ❖ Like fopen, we should always check if malloc manages to get the memory.

```
float *farray;  
/* Try to allocate memory for 1000 floats */  
farray= malloc(1000*sizeof(float));  
if (farray == NULL) {  
    fprintf (stderr, "Out of memory!\n");  
    exit (-1);  
}
```

C++ Input and Output

- ❖ Variety of I/O functions in *C++ iostream Library*.
- ❖ Must include `<iostream.h>` to use them.
- ❖ Must use `g++` to compile

❖ `cout << "Hello world\n";`

❖ `cin >> x;`

» Read into variable `x`;