

File Handling

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) {  
    printf ("ERROR IN FILE CREATION");  
    /* 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"** creates 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
- **"rb"** read binary file (raw bytes)
- **"wb"** write binary 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

Usage of exit()

```
FILE *fptr;  
char filename[] = "file2.dat";  
fptr = fopen (filename, "w");  
if (fptr == NULL) {  
    printf ("ERROR IN FILE CREATION");  
    /* Do something */  
    exit(-1);  
}
```

Writing to a file using fprintf()

- **fprintf()** works just like printf and sprintf except that its first argument is a file pointer.

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

Reading Data Using fscanf()

- We also read data from a file using fscanf().

```
FILE *fptr;
```

```
fptr= fopen (“input.dat”,“r”);
```

```
/* Check it's open */
```

```
if (fptr==NULL)
```

```
{
```

```
    printf(“Error in opening file \n”);
```

```
}
```

```
fscanf(fptr,“%d%d”,&x,&y);
```

input.dat



20 30



x=20

y=30

Reading lines from a file using `fgets()`

We can read 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.

```
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);
```

Opening

Access

closing

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");
```

An example program

```
#include <stdio.h>
int main()
{
    int i;
```

Give value of i

15

Value of i=15

No error: But an example to show error message.

```
    fprintf(stdout, "Give value of i \n");
    fscanf(stdin, "%d", &i);
    fprintf(stdout, "Value of i=%d \n", i);
    fprintf(stderr, "No error: But an example to show error message.\n");
}
```

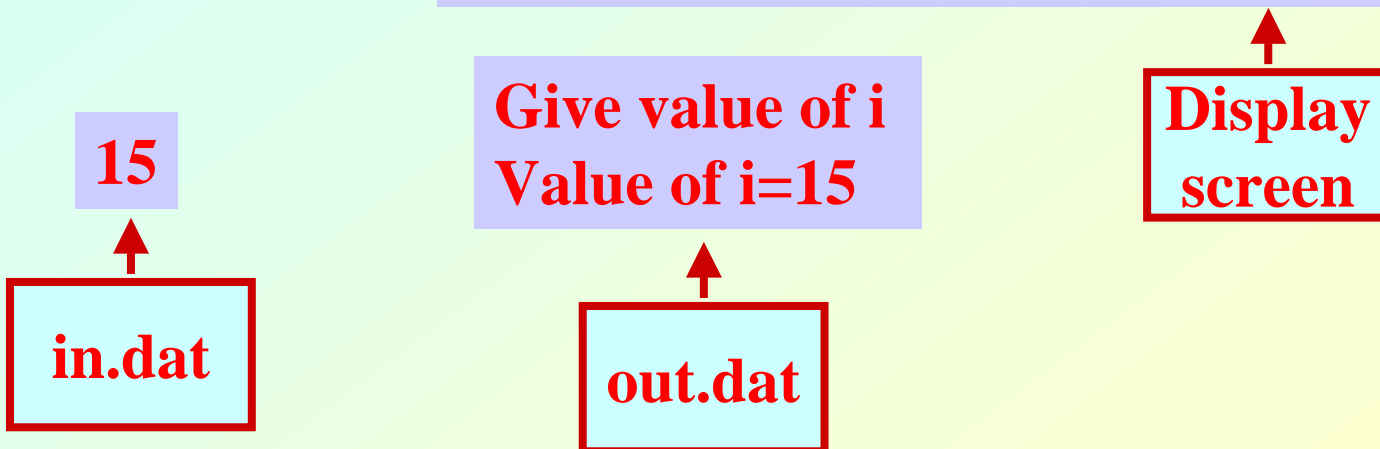
**Display on
The screen**

Input File & Output File redirection

- One may redirect the input and output files to other files (other than **stdin** and **stdout**).
- Usage: Suppose the executable file is **a.out**

```
$ ./a.out <in.dat >out.dat
```

No error: But an example to show error message.



Reading and Writing a character

- A character reading/writing is equivalent to reading/writing a byte.

```
int getchar( );
```

```
int fgetc(FILE *fp);
```

```
int putchar(int c);
```

```
int fputc(int c, FILE *fp);
```

- **Example:**

```
char c;
```

```
c=getchar( );
```

```
putchar(c);
```

Example: use of getchar() and putchar()

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
int c;
```

```
printf("Type text and press return to see it again \n");
```

```
printf("For exiting press <CTRL D> \n");
```

```
while((c=getchar( ))!=EOF) putchar(c);
```

```
}
```



End of file

Command Line Arguments

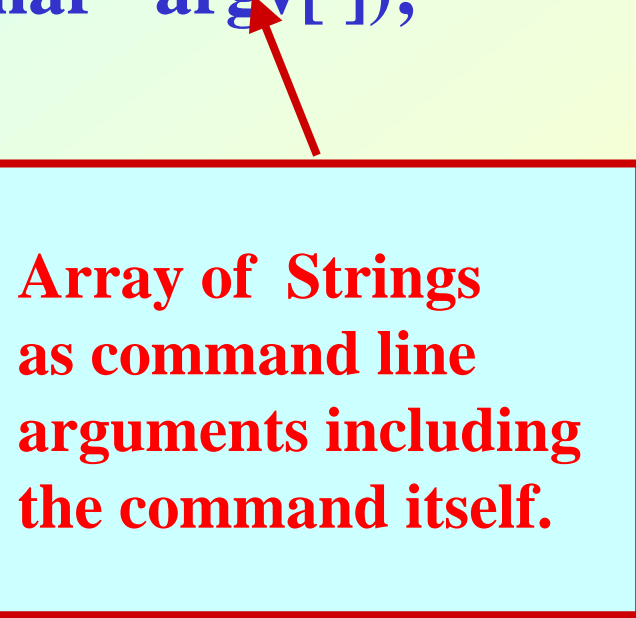
- Command line arguments may be passed by specifying them under `main()`.

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

**Argument
Count**



**Array of Strings
as command line
arguments including
the command itself.**



Example: Reading command line arguments

```
#include <stdio.h>
#include <string.h>

int main(int argc,char *argv[])
{
FILE *ifp,*ofp;
int i,c;
char src_file[100],dst_file[100];

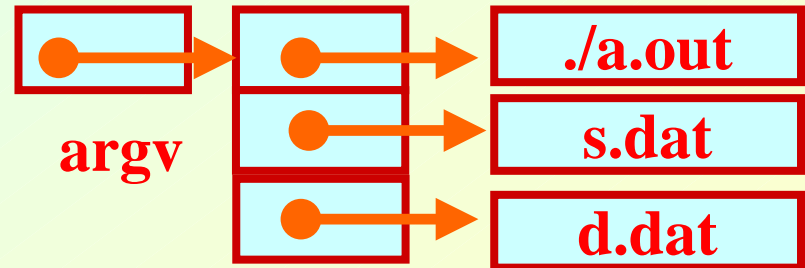
if(argc!=3){
printf("Usage: ./a.out <src_file> <dst_file> \n");
exit(0);
}
else{
strcpy(src_file,argv[1]);
strcpy(dst_file,argv[2]);
}
```


Example: Contd.

```
if((ifp=fopen(src_file,"r"))==NULL)
{
    printf("File does not exist.\n");
    exit(0);
}
if((ofp=fopen(dst_file,"w"))==NULL)
{
    printf("File not created.\n");
    exit(0);
}
while((c=getc(ifp))!=EOF){
    putc(c,ofp);
}
fclose(ifp);
fclose(ofp);
}
```

./a.out s.dat d.dat

argc=3



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

Example: Averaging from Command Line

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
$ ./a.out 45 239 123
```

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

```
{
```

```
    float sum=0;
```

```
    int i,num;
```

```
    num=argc-1;
```

```
    for(i=1;i<=num;i++)
```

```
        sum+=atof(argv[i]);
```

```
    printf("Average=%f \n",sum/(float) num);
```

```
}
```

```
Average=135.666667
```