241 lecture note: For week7

Command line arguments are parameters supplied to a program, when the program is invoked.

main can take 2 arguments, conventionally called argc and argv.

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

argc

– Number of arguments (including program name)

argv

– Array of char\*s (that is, an array of ‘c’ strings)

– argv[0] à program name

– argv[1] à first argument

– …

– argv[argc-1] à last argument

$ ./main\_arg NWEN241 is about Systems Programming using C 8 arguments

0: ./main\_arg

1: NWEN241

2: is

3: about

4: Systems

5: Programming

6: using

7: C

$

Total of 8 arguments including program name itself. Arguments are read in as strings.

In general, I/O is the process of copying data between main memory and external devices

In C, everything is a file; --🡪each file is simply a sequential stream of bytes;

C imposes no structure on a file.

BUT, Defined in stdio.h is the struct FILE that comprises a file descriptor and a file control block

A file must first be opened properly before it can be accessed for reading or writing.

When a file is opened, a stream is associated with the file. Pointer to (i.e. address of) the “file” is returned

Input / Output & stdio.h

Every UNIX/Linux process begins with three open files corresponding to the standard input, output and error streams, macros defined in stdio.h:

****

Also defined in stdio.h are three variable types (including FILE), several macros (including above) and various functions for performing input / output

e.g. printf(), scanf(), getchar() , gets(), putchar(), puts(), etc.

File operations

1. Creating a new file
2. Opening an existing file
3. Writing data to a file
4. Reading data from a file
5. Closing a file
6. Random access operations

Declaring FILE pointer and Opening file

A file must be “opened” before it can be used.

FILE \*fp; // pointer to data type FILE

fp = fopen (filename, mode);

fopen 🡪returns a pointer (fp) to the file;

* used in all subsequent file operations.

mode 🡪 “r” – open the file for reading only

“w” – open the file for writing only

“a” – open the file for appending data to it

Did the fopen(…) command succeed?

If the file was not able to be opened, then the value returned by the fopen routine is NULL.

For example, if the file mydata does not exist, then:

**FILE \*fptr ;**

**fptr = fopen ("mydata", "r") ;**

**if (fptr == NULL) {**

**printf ("File open failed.\n");**

**}**

Closing a file

After completing all operations on a file, it must be closed to ensure that all file data stored in memory buffers are written to the file.

**General format:** fclose (file\_pointer);

FILE \*fp; // pointer to data type FILE

:::

fp = fopen (filename, mode);

:::

fclose (fp); // close the file

Read/Write Operations on Files

Simplest file input-output (I/O) function: **getc & putc**

**char ch;**

**FILE \*fp;**

**:::**

**ch = getc(fp);**

**getc** will return an end-of-file marker EOF, when the

end of the file has been reached.

**putc** is used to write a character to a file.

**char ch;**

**FILE \*fp;**

**:::**

**putc(c, fp);**

**main() {**

**FILE \*ifp, \*ofp;**

**char c;**

**ifp = fopen ("ifile.dat" , " r ");**

**ofp = fopen ("ofile.dat" , " w ");**

**while ((c = getc (ifp)) != EOF)**

**putc (toupper(c), ofp);**

**fclose (ifp);**

**fclose (ofp);**

**}**

1. fgetc() vs getc()

• **fgetc** is a subroutine that performs the same function as the **getc** macro; **fgetc** is NOT a macro.

• **fgetc** subroutine runs more slowly than **getc** but takes less disk space.

• Benefit: fgetc(\*p++) works but getc(\*p++) fails

1. fputc() vs putc()

• fputc is a subroutine while putc is a macro;

• same considerations for fputc as fgetc.

fscanf()

Same as scanf except need to **file pointer** as an argument.

Example:

int a, b;

FILE \*fptr1;

fptr1 = fopen ("datafile", "r");

fscanf( fptr1, "%d%d", &a, &b);

fscanf would read values from the file "pointed"

to by **fptr1** and **assign those values** to **a** and **b**.

End of File using EOF

The end-of-file indicator **EOF** informs the program when there are no more data (no more bytes) to be processed.

Check the value returned by the **fscanf** function:

**int istatus, var;**

**istatus = fscanf (fptr1, "%d", &var) ;**

**if ( istatus == EOF ) {**

**printf ("End-of-file encountered.\n") ;**

**}**