

# ESC101: Introduction to Computina

Command Line  
&  
File Handling



# Arguments on the Command Line

◆ Typically when using commands we provide arguments to the command in the same line.

- `cd my_dir`
- `gcc my_file.c`
- `cp file1.c file2.c`

◆ In each case, stuff in red is the command line argument

◆ In the third example, `cp` is the command name and `file1.c` and `file2.c` are its two arguments.

# Command Line Args in C

◆ Write a program to read a name from command line, and say "Hello" to it.

◆ Some Example Interaction (Output in red):

```
$ ./a.out ABC
```

```
Hello ABC
```

```
$ ./a.out World
```

```
Hello World
```

```
$ ./a.out ESC101
```

```
Hello ESC101
```

Note that the program really has no sense of what is a name. It just prints the argument provided.

# Command Line Args = Args to main

◆ So far we used the following signature for main

`int main()`

◆ But main can take arguments. The modified prototype of main is

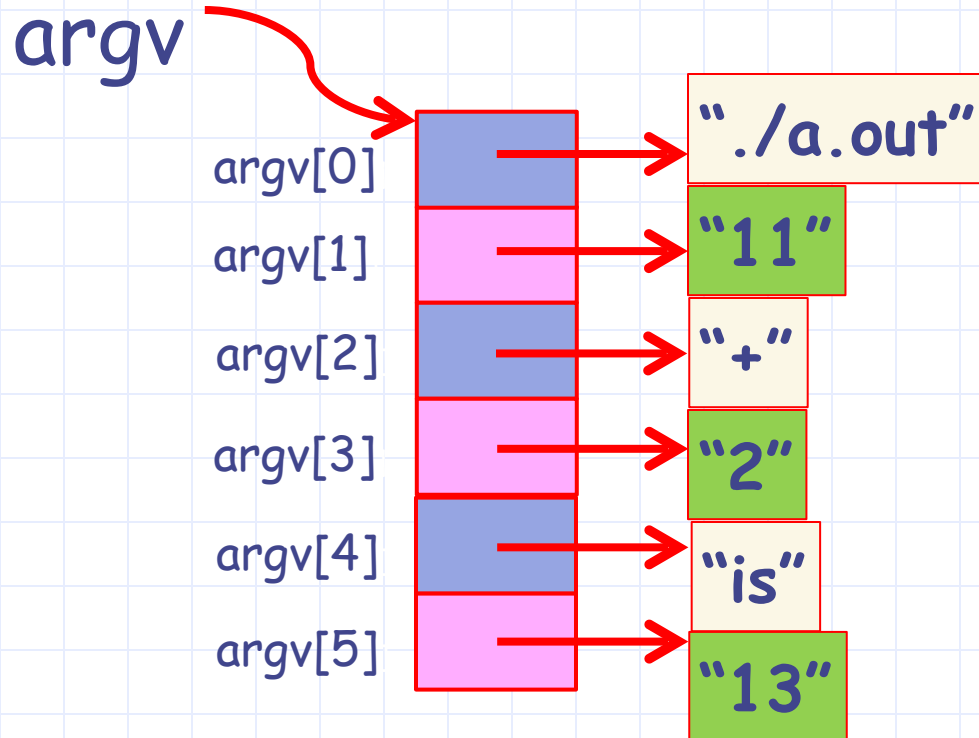
`int main(int argc, char **argv)`

- Argument Count (`argc`): An int that tells the number of arguments passed on command line
- Argument Values (`argv`): Array of strings. `argv[i]` is the i-th argument as string.

# Args to main

`./a.out 11 + 2 is 13`

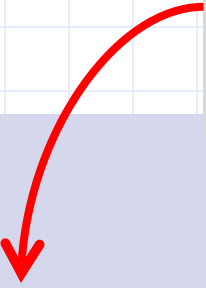
`argc = 6`     `./a.out` is included in arguments



Note that everything is treated as string, even the numbers!

# Example

**NOTE:** char \*\*argv is same as char \*argv[]



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

```
int main(int argc, char *argv[]) {  
    if (argc < 2)  
        printf ("Too few args!\n");  
    else if (argc == 2)  
        printf ("Hello %s\n", argv[1]);  
    else  
        printf ("Too many args!\n");  
    return 0;  
}
```

```
$ ./a.out
```

Too few args!

```
$ ./a.out ABC
```

Hello ABC

```
$ ./a.out World
```

Hello World

```
$ ./a.out ESC101
```

Hello ESC101

```
$ ./a.out Hey There
```

Too many args!

# What about Other Types?

◆ Write a program that takes two numbers (integers) on command line and prints their sum.

◆ Problem:

- Everything on command line is read as string!
- How do I convert string to int?

◆ Solution: Library functions in `stdlib.h`

- **atoi**: takes a string and converts to int

`atoi("1234")` is 1234, `atoi("123ab")` is 123, `atoi("ab")` is 0

- **atof**: converts a string to double

◆ Other variations : **atol**, **atoll**

# Adding 2 Numbers

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[]) {
    if (argc != 3)
        printf ("Bad args!\n");
    else {
        int a = atoi(argv[1]);
        int b = atoi(argv[2]);
        printf ("%d\n",a+b);
    }
    return 0;
}
```

```
$ ./a.out
```

Bad args!

```
$ ./a.out 3 4
```

7

```
$ ./a.out 3 -4
```

-1

```
$ ./a.out 3 four
```

3

```
$ ./a.out 3 4 5
```

Bad args!



# Command Line Sorting

```
int main(int argc, char *argv[]) {
    int *ar, n;
    n = argc - 1;
    ar = (int *)malloc(sizeof(int) * n);
    for (i=0; i<n; i++)
        ar[i] = atoi(argv[i+1]);

    merge_sort(ar, n); // or any other sort

    for (i=0; i<n; i++)
        printf("%d ", ar[i]);
    return 0;
}
```

```
void merge_sort (
    int *arr, int n)
{
    ...
}
```

```
$ ./a.out 1 4 2 5 3 9 -1 6 -10 10
-10 -1 1 2 3 4 5 6 9 10
```

# 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), and of course, files that a user creates (`/home/don/input.txt`, `/home/don/Esc101/lab12.c`)

# 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
  - and so on.

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

## 2. Read/Write to the file

```
int fscanf(FILE *fp, char *format, ...)  
int fprintf(FILE *fp, char *format, ...)
```

## 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;  
}
```



```
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.
  - ❖ returns a value of zero if the operation was successful, and a nonzero value to indicate failure
- `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 something", 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

# An Exercise

- Often, events in a system are logged on to a particular file. (e.g. usb drive mounted, user logs off etc.)
- These log files can be quite large. We are usually interested in the latest events (maybe the last 10 events.)
- The unix command "tail <filename>" prints the last 10 lines of <filename>. Can you program this?
- (Hint: Start at end of file, and use fseek.)

“Computer science is not about machines, in the same way that astronomy is not about telescopes. There is an essential unity of mathematics and computer science.”

-- 1990s Folklore. Sometimes attributed to Edsger W. Dijkstra.