Command Line Argument

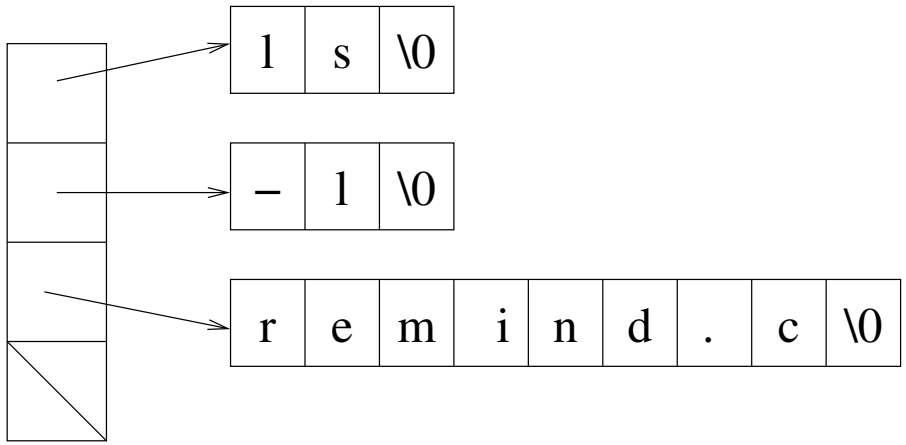
Parameter List in Main

- Sometime additional information (switch) may have to be supplied to program.
- Eg. ls -l uses parameter l for changing its default behavior.
- Similarly, ls -l remind.c: uses parameters l and reminder.c to change its default behavior.
- To access command line information in function main two parameters are added: int main(int argc, char *argv[])
- argc: argument count.
- argv[0]: is the name of the program
- argv[1] ... arg[argc-1]: are different switches to program.

Command Line Argument

Parameter List in Main

- For eg., ls -l reminder.c, has argument count 3.
- argv[0] points to string ls
- argv[1] points to string -l
- argv[2] points to string remind.c
- argv[3] is NULL



Command Line Argument

Example

```
#include <stdio.h>
#include <string.h>
#define NPLANETS 9
int main(int argc, char * argv[]) {
    char *planets[] = {"Mercury", "Venus", "Earth",
                       "Mars", "Jupiter", "Saturn",
                       "Uranus", "Neptune", "Pluto"};
    int i, j;

    /*** Remaining part of the code ***/
}
```

Command Line Argument

Example

```
for(i = 1; i < argc; i++) {
    for(j = 0; j < NPLANETS; j++)
        if(strcmp(argv[i], planets[j]) == 0) {
            printf("%s is planet No. %d\n", argv[i], j + 1);
            break;
        }
    if (j == NPLANETS)
        printf("%s is not a planet\n", argv[i]);
}
```

Handling Text Files in C

Streams

- **Stream**: any source of input or any destination for output.
- So far only one stream was used for each, namely, **keyboard** for all input, and **screen** for output.
- Programs may need additional streams often represented by files stored in HDD, CD/DVD, etc.
- Also represented by network ports, printer etc which don't store files.
- Let us talk about files (which alternate for streams) only.
- Functions in **stdio.h** work equally well for any stream not just files.

Handling Text Files in C

File Pointers

- Accessing a stream is done through a file pointer.
- Its type is `FILE *` which is declared in `stdio.h`
- Certain streams are represented by file pointers with standard names.
- For other file pointers should be declared: `FILE *fp1, *fp2;`
- Standard file pointers are `stdin, stdout, stderr`
- We neither have to open nor have to close these pointers.

Handling Text Files in C

Parameter List in Main

- Standard streams can be redirected to get these represented by files associated with other devices.
- Input redirection forces the input to be read from a file.
- Similarly output redirection forces the output to be sent to file.

Example

For example,
program `<in.dat >out.dat`
takes input from `in.dat` and throws output to `out.dat`

Handling Text Files in C

Text Files

- `<stdio.h>` supports both binary and text files.
- Text files have following characteristics:
  - Divided into lines, each terminated by a linefeed character.
  - May contain a special `EOF` (`CTRL-Z`), but this not required in Linux.
- Binary files do not have `EOL` or `EOF`, all bytes are treated equally.
- Bytes will be reversed in m/c that store data in little endian order.
- When program reads/write data from/to a file, we need to take into account whether it is a binary/text file.

Handling Text Files in C

Need for Binary File

- A program that displays content of a file onto screen will use a text file.
- But a file copying program can not assume file to be copied as a text file, because on encountering `EOF` rest of the file will be ignored.
- `EOF` may be just a valid item in the file being copied.
- So it is safer to assume file to be a binary file.

Handling Text Files in C

File Operations

- Opening a file: `fopen("File Name", "mode");`
  - `fopen`: returns a file pointer which must be saved for further operations (read/write).
  - `File Name`: could be complete with full/relative path.
  - `Mode`: read (`"r"`) or write (`"w"`) or read/write (`"rw"`).
- Closing a file: `fclose(fileptr);` where `fileptr` is obtained from an `fopen` or `freopen`.

Handling Text Files in C

Modes

String	Description
"r"	Reading
"w"	Writing, file need not exist
"a"	Append, file need not exist
"r+"	Reading and writing from beginning
"w+"	Reading and writing (truncate if file exsist)
"a+"	Reading and writing (append if file exsist)

Handling Text Files in C

Modes for Binary Files

String	Description
"rb"	Reading
"wb"	Writing, file need not exist
"ab"	Append, file need not exist
"r+b" /"rb+"	Reading and writing from beginning
"w+b" /"wb+"	Reading and writing (truncate if file exsist)
"a+b" /"ab+"	Reading and writing (append if file exsist)

Handling Text Files in C

Example

```
#include <stdio.h>
#include <stdlib.h>
#define FILE_NAME "example.dat"
int main() {
    FILE *fptr; // Declare a file pointer
    fptr = fopen(FILE_NAME, "r"); // Save the file pointer
    if (fptr == NULL) {
        printf("Can not open %s\n", FILE_NAME);
        exit(EXIT_FAILURE);
    }
    ...

    fclose(fptr); // Close the file
}
```

Handling Text Files in C

Attaching a File to an Open Stream

- `freopen` attaches a different file to a stream that is already open.
- Most common use is to attach standard streams: `stdin`, `stdout`, `stderr`.
- Eg: `freopen("myfile", "w", stdout);` causes stdout to be represented by `myfile`.
- It closes any file previously associated with stdout then reopens the same by associating it with `myfile`.

Handling Text Files in C

Example

```
include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    FILE *fp;

    if (argc != 2) {
        printf("Usage: can open filename\n");
        exit(EXIT_FAILURE);
    }
    if ((fp = fopen(argv[1], "r")) == NULL) {
        printf("%s can't be opened\n", argv[1]);
        exit(EXIT_FAILURE);
    }
    printf("%s can be opened\n", argv[1]);
    fclose(fp);
}
```

Advanced File Operations

Example

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    FILE *fpsrc, *fpdest;
    char ch;

        /*** Code for errors in arguments/opening ***/

    while ((ch = getc(fpsrc)) != EOF)
        putc(ch, fpdest);
    fclose(fpdest);
    fclose(fpsrc);
}
```

Advanced File Operations

Example

```
if (argc != 3) {
    fprintf(stderr, "Usage: fcopy src dest\n");
    exit(EXIT_FAILURE);
}
if ((fpsrc = fopen(argv[1], "rb")) == NULL) {
    printf("%s can't be opened\n", argv[1]);
    exit(EXIT_FAILURE);
}
if ((fpdest = fopen(argv[2], "wb")) == NULL) {
    printf("%s can't be opened\n", argv[2]);
    exit(EXIT_FAILURE);
}
```