

# The C Standard Library Part 3

`strtol` and Stream-based File I/O

strtol


# The Problem with `atoi`

- How to detect malformed strings?

# The Problem with `atoi`

- How to detect malformed strings?


```
int main(int argc, char **argv) {  
    if(argc != 2) {  
        printf("usage: %s <number>\n", argv[0]);  
        return -1;  
    }  
  
    int n = atoi(argv[1]);  
    printf("n is: %d\n");  
    return 0;  
}
```

[13-standard-library-3/atoi-limitation.c](#) 

# The Problem with `atoi`

- How to detect malformed strings?

```
int main(int argc, char **argv) {  
    if(argc != 2) {  
        printf("usage: %s <number>\n", argv[0]);  
        return -1;  
    }  
  
    int n = atoi(argv[1]);  
    printf("n is: %d\n");  
    return 0;  
}
```

[13-standard-library-3/atoi-limitation.c](#) 

**`atoi` does not offer any way to know the string is invalid or overflows/underflows an `int`!**

# Solution: Use `strtol`

```
long strtol(const char *nptr, char **endptr, int base);
```

[man](#)

# Solution: Use `strtol`

```
long strtol(const char *nptr, char **endptr, int base);
```

[man](#)

- Convert the string `nptr` into a long integer and return that integer
- `base` can be 10, 8, 16, etc.

# Solution: Use `strtol`

```
long strtol(const char *nptr, char **endptr, int base);
```

[man](#)

- Convert the string `nptr` into a long integer and return that integer
- `base` can be 10, 8, 16, etc.
- after the call `*endptr` points to the first invalid character of the string, and to `'\0'` if the string is fully valid



# Solution: Use `strtol`

```
long strtol(const char *nptr, char **endptr, int base);
```

[man](#)

- Convert the string `nptr` into a long integer and return that integer
- `base` can be 10, 8, 16, etc.
- after the call `*endptr` points to the first invalid character of the string, and to `'\0'` if the string is fully valid
- Under/overflows cause `errno` to be set to `ERANGE`, and the function returns `LONG_MIN` (underflow) or `LONG_MAX` (overflow)

# strtol Usage Example

```
/* ... */
#include <errno.h>
#include <limits.h>

int main(int argc, char **argv) {
    if(argc != 2) { /* ... */ }

    char *endptr;
    long n = strtol(argv[1], &endptr, 10);

    if(*endptr != '\0') {
        printf("invalid string!\n");
        return -1;
    }

    if(errno == ERANGE) {
        if(n == LONG_MIN) printf("underflow!\n");
        if(n == LONG_MAX) printf("overflow!\n");
        return -1;
    }

    printf("n is: %ld\n", n);
    return 0;
}
```

[13-standard-library-3/strtol.c](#) 

# strtol Usage Example

```
/* ... */
#include <errno.h>
#include <limits.h>


int main(int argc, char **argv) {
    if(argc != 2) { /* ... */ }

    char *endptr;
    long n = strtol(argv[1], &endptr, 10);

    if(*endptr != '\0') {
        printf("invalid string!\n");
        return -1;
    }

    if(errno == ERANGE) {
        if(n == LONG_MIN) printf("underflow!\n");
        if(n == LONG_MAX) printf("overflow!\n");
        return -1;
    }

    printf("n is: %ld\n", n);
    return 0;
}
```

[13-standard-library-3/strtol.c](#) 

- Check string validity by looking at the value of `*endptr`

# strtol Usage Example

```
/* ... */
#include <errno.h>
#include <limits.h>


int main(int argc, char **argv) {
    if(argc != 2) { /* ... */ }

    char *endptr;
    long n = strtol(argv[1], &endptr, 10);

    if(*endptr != '\0') {
        printf("invalid string!\n");
        return -1;
    }

    if(errno == ERANGE) {
        if(n == LONG_MIN) printf("underflow!\n");
        if(n == LONG_MAX) printf("overflow!\n");
        return -1;
    }

    printf("n is: %ld\n", n);
    return 0;
}
```

[13-standard-library-3/strtol.c](#) 

- Check string validity by looking at the value of `*endptr`
- Check for under/overflows with `errno` and the return value

# Stream-based File I/O

# Stream-based File I/O

```
FILE *fopen(const char *pathname, const char *mode);
```

[man](#)

- Opens the file identified by `pathname` and return a stream object (`FILE *` data structure), returns `NULL` on error

# Stream-based File I/O

```
FILE *fopen(const char *pathname, const char *mode);
```

[man](#)

- Opens the file identified by `pathname` and return a stream object (`FILE *` data structure), returns `NULL` on error
- `mode` can be:
  - `"r"`: read-only
  - `"r+"`: read-write
  - `"w"`: write-only, truncate file if it exists, create it if it does not
  - `"w+"`: read-write, truncate file if it exists, create it if it does not
  - and more, see man page

# Stream-based File I/O

```
FILE *fopen(const char *pathname, const char *mode);
```

[man](#)

- Opens the file identified by `pathname` and return a stream object (`FILE *` data structure), returns `NULL` on error
- `mode` can be:
  - `"r"`: read-only
  - `"r+"`: read-write
  - `"w"`: write-only, truncate file if it exists, create it if it does not
  - `"w+"`: read-write, truncate file if it exists, create it if it does not
  - and more, see man page
- Close stream with `fclose`:

```
int fclose(FILE *stream);
```

[man](#)



# Stream-based File I/O

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);  
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

[man](#)

# Stream-based File I/O

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);  
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

[man](#)

- Read/write `nmemb` contiguous items, of size `size` each, from/to the file `stream`, to/from the memory pointed by `ptr`

# Stream-based File I/O

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);  
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

[man](#)

- Read/write `nmemb` contiguous items, of size `size` each, from/to the file `stream`, to/from the memory pointed by `ptr`
- Return the total number of **items** read/written
  - I.e. total number of bytes transferred only when `size` is 1

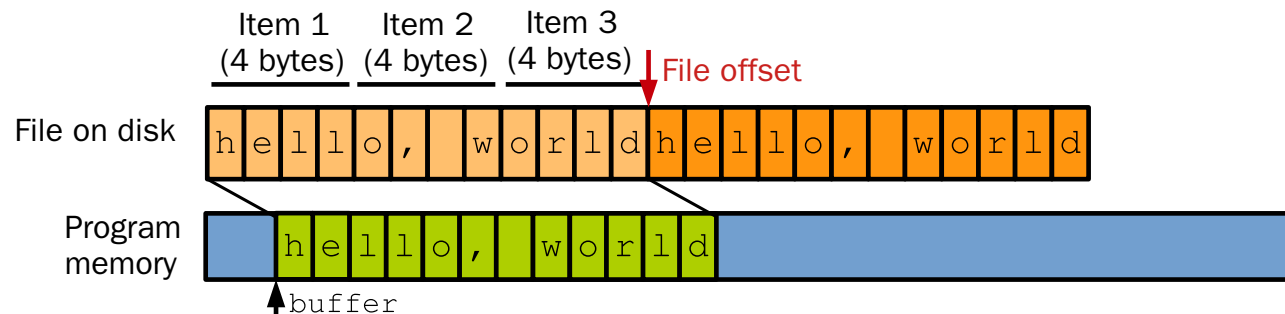
# Stream-based File I/O

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);  
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

[man](#)

- Read/write **nmemb** contiguous items, of size **size** each, from/to the file **stream**, to/from the memory pointed by **ptr**
- Return the total number of **items** read/written
  - I.e. total number of bytes transferred only when **size** is 1
- Example: reading 3 items of size 4 bytes:

```
items_read = fread(buffer, 4, 3, file_ptr);
```



# Stream-based File I/O: Example

```
#include <stdio.h>

char *alphabet = "abcdefghijklmnopqrstuvwxyz";

int main(int argc, char **argv) {
    FILE *f1, *f2;
    char buffer[27];

    f1 = fopen("test-file.txt", "w");
    if(f1 == NULL) {
        perror("fopen");
        return -1;
    }

    if(fwrite(alphabet, 2, 13, f1) != 13) {
        perror("fwrite");
        fclose(f1);
        return -1;
    }

    fclose(f1);
```

```
    f2 = fopen("test-file.txt", "r");
    if(f2 == NULL) {
        perror("fopen");
        return -1;
    }

    if(fread(buffer, 1, 26, f2) != 26) {
        perror("fread");
        fclose(f2);
        return -1;
    }

    buffer[26] = '\0';
    printf("read: %s\n", buffer);

    fclose(f2);
    return 0;
}
```

//

[13-standard-library-3/file-stream.c](#) 

# Summary

- `strtol` to convert strings to integers in a robust way
  - `FILE` \* I/O, higher level operations than `read/write/etc.`
- 

Feedback form: <https://bit.ly/3R8dSGr>

