High Performance Computing

Day 4 - Project 1

Task

- Implement a fetch_line( ) function in C that does the following :

  1. Retrieve next available nonempty line from an input stream

  2. After reading a line, it should trim the line of comments and leading and trailing space,

  3. If what remains is a string of zero length, it should go on to read the next line and repeat the process; otherwise, return a pointer to the first character of the trimmed string.

  4. Upon the end of the input or an error condition, it returns NULL

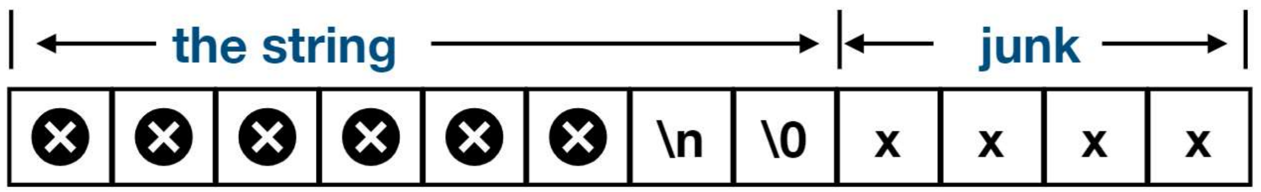
Example: using fgets

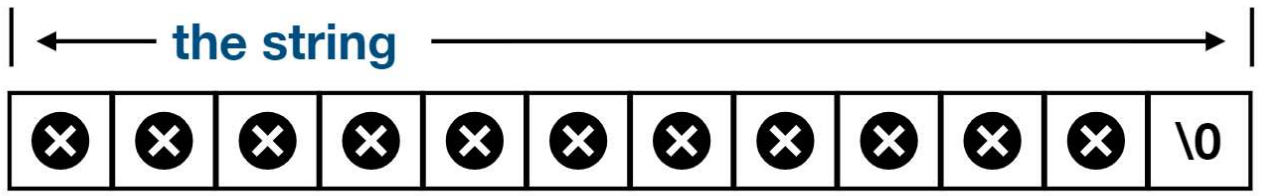
fgets reads at most 11 characters
and stores them in buf.
Reading stops when:

1. It encounters \n
2. It reaches the limit of 11 characters
3. The input stream ends or an error occurs

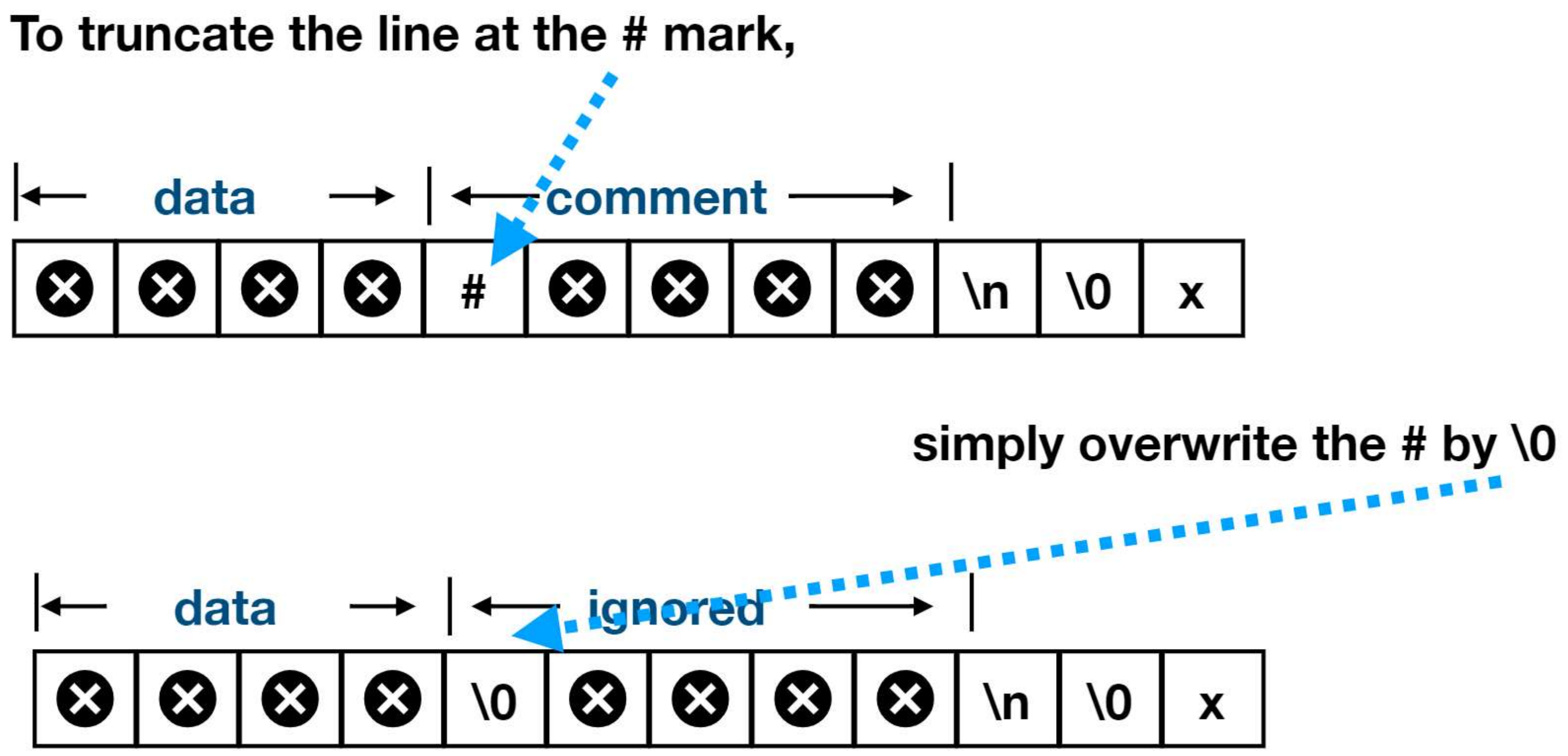
fgets_demo.c

```
#include <stdio.h>
#define BUFLEN 12
int main (void)
{
  char buf[BUFLEN];
  while (fgets(buf, BUFLEN, stdin) != NULL)
    printf("%s", buf);
  return 0;
}
```





trim white space & comments


To truncate the line at the # mark,
data
comment
#
\n
\0
x
simply overwrite the # by \0
data
ignored
\0
\n
\0
x

fetch_line_demo

fetch_line_demo.c

```c
#include <stdio.h>
#include "fetch_line.h"
#define BUFLEN 40
int main(void)
   {
     char buf[BUFLEN];
     char *s;
     int lineno = 0;
     while((s=fetch_line(buf, BUFLEN, stdin, &lineno)) != NULL)
      printf("trimmed line %3d: %s\n", lineno, s);
    return 0;
  }
```

```
gcc -c -Wall -pedantic -std=c89 -O2 fetch_line_demo.c
gcc  fetch_line.o fetch_line_demo.o -o fetch-line-demo
```

fetch_line.c

fetch_line.c

```c
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include "fetch_line.h"

static char *trim_line(char *s);
char *fetch_line(char *buf, int buflen, FILE *stream, int *lineno);

char *fetch_line(char *buf, int buflen, FILE *stream, int *lineno)
{
  char *s;
  if (fgets(buf, buflen, stream) == NULL)
    return NULL;
  ++*lineno;
  if (buf[strlen(buf) -1] != '\n') {
    fprintf(stderr, "*** reading error: input line %d too"
            "long for %s's buf[%d]\n",
            *lineno, __func__, buflen);
    exit(EXIT_FAILURE);
    }
 s = trim_line(buf);
 if (*s != '\0')
    return s;
 else
   return fetch_line(buf, buflen, stream, lineno);
}
```

gcc -c -Wall -pedantic -std=c89 -O2 fetch_line.c

Please implement this function (trim_line) by yourselves

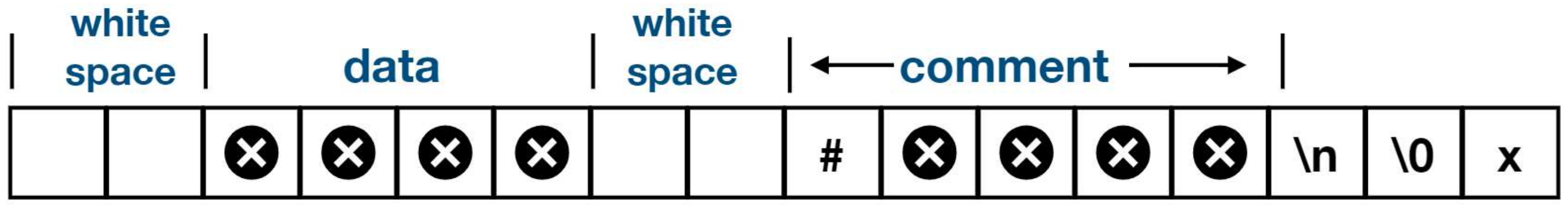
fetch_line.h

fetch_line.h

```
#ifndef H_FETCH_LINE_H
#define H_FETCH_LINE_H
#include <stdio.h>
char *fetch_line(char *buf, int buflen, FILE *stream, int *lineno);
#endif /* H_FETCH_LINE_H */
```

implementing trim_line - 1

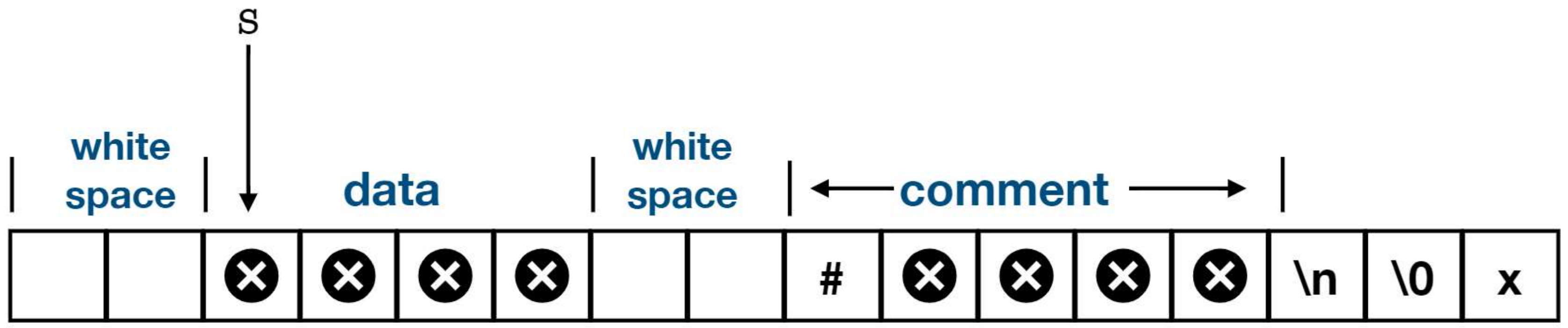
Let us discuss a strategy for trim_line using the sample line shown below



implementing trim_line - 2

Let us discuss a strategy for trim_line using the sample line shown below

1. **move a pointer `s` forward, one character at a time, skipping over whitespaces, and stopping at the first non whitespace character**

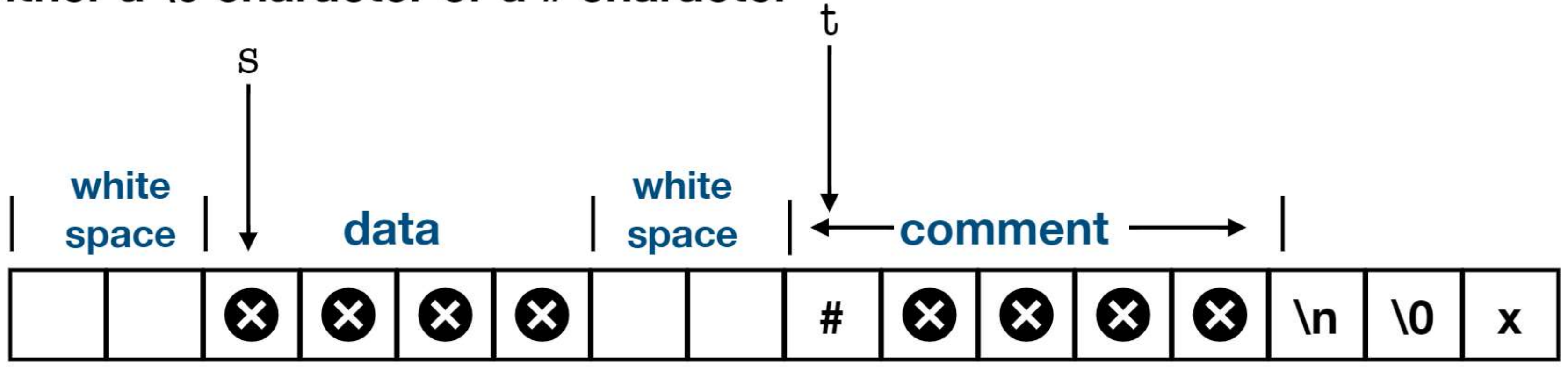


2. **by positioning the pointer here, we have effectively trimmed the** *leading white space*

implementing trim_line - 3

Let us discuss a strategy for trim_line using the sample line shown below

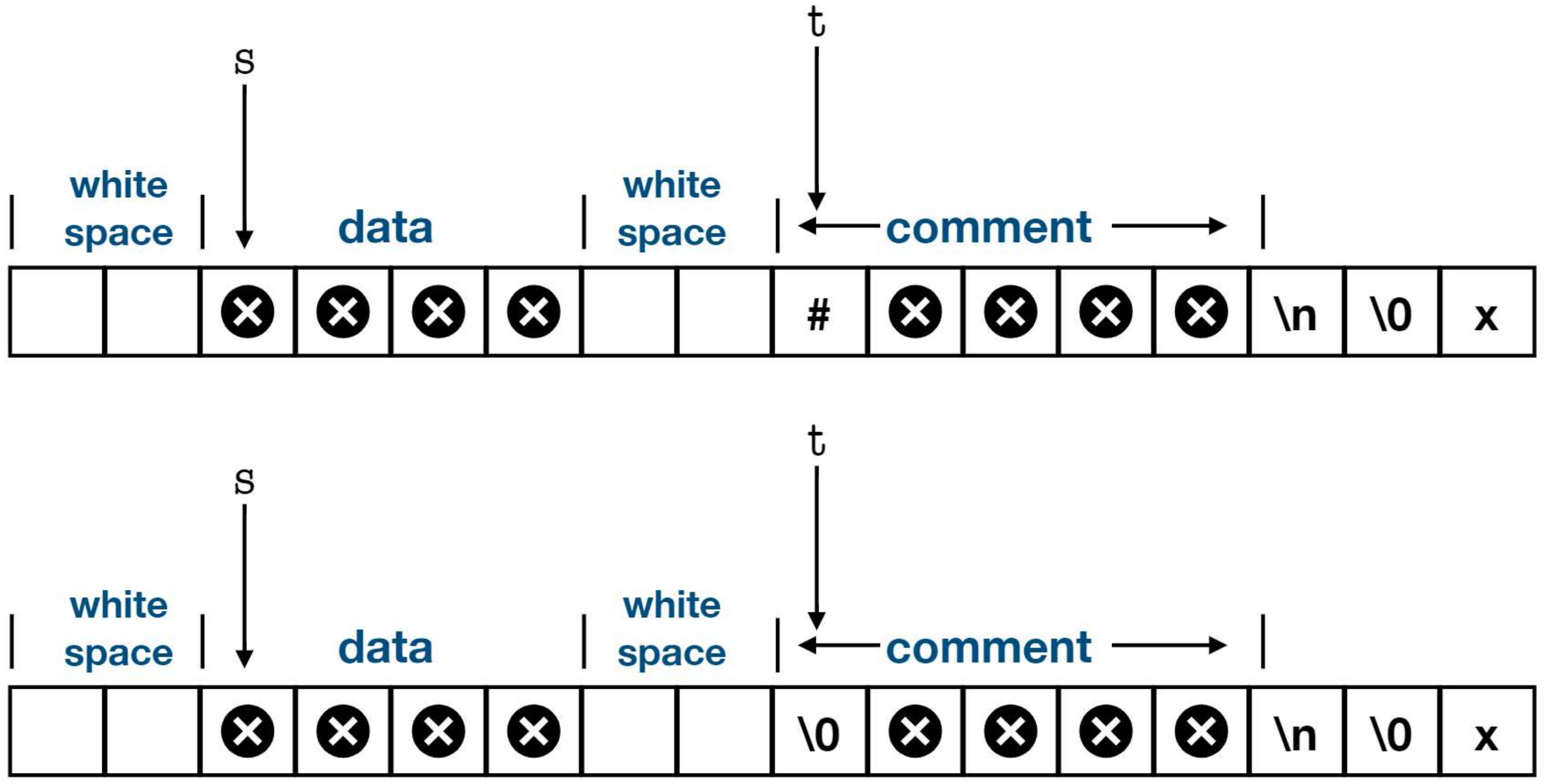
3. Set a second pointer, t, initially at s, and then move it forward until you arrive at either a \0 character or a # character



implementing trim_line - 4

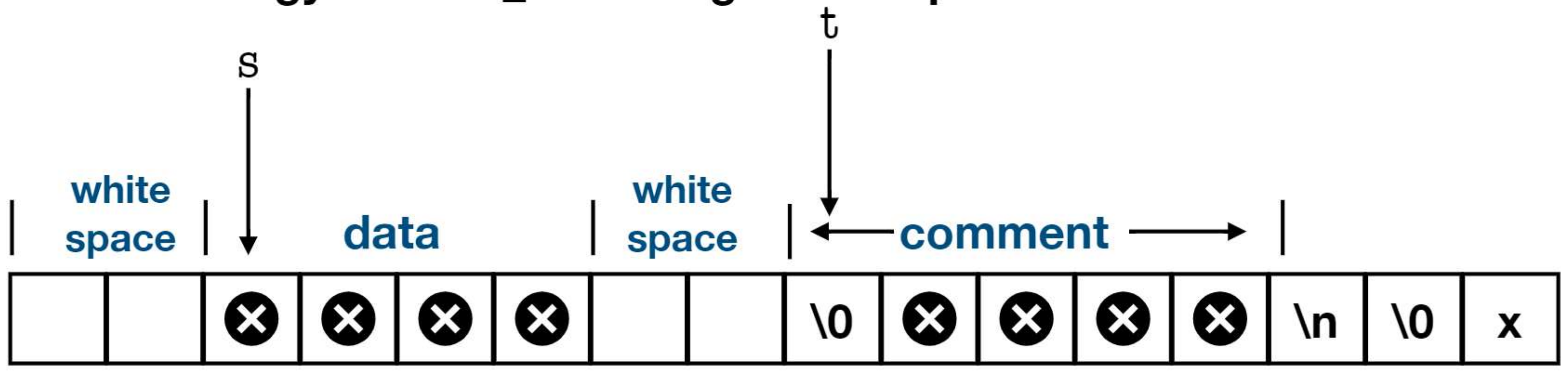
Let us discuss a strategy for trim_line using the sample line shown below

4. since we found a # character, let us change it to \0

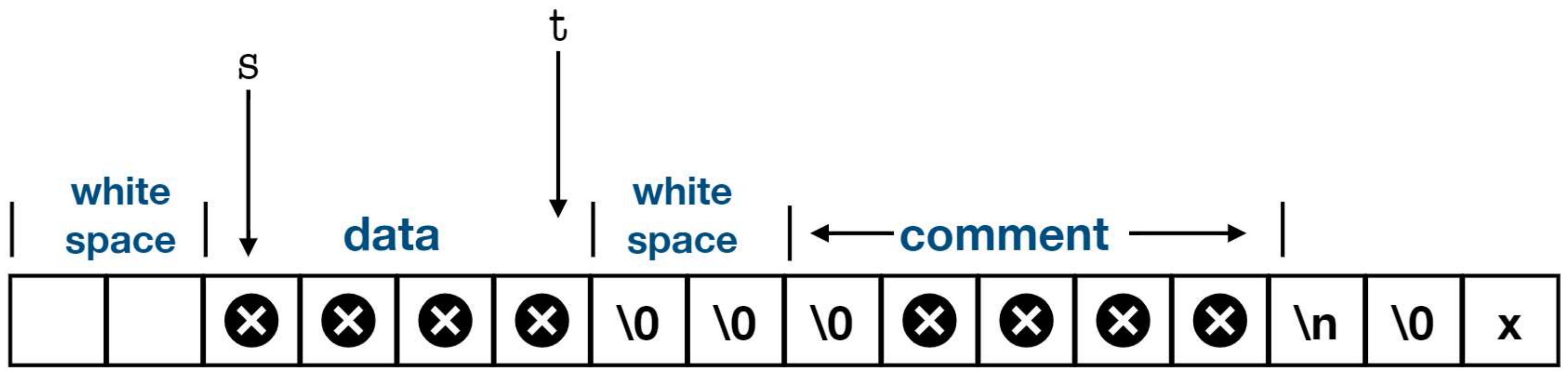


implementing trim_line - 5

Let us discuss a strategy for trim_line using the sample line shown below



5. If position t is different from s, move t backward, one character at a time, overwriting any whitespace with a \0 till you stop at the first non whitespace character



At the end of step 5, the pointer s points to the trimmed string, and you can return the pointer to the caller.

Things to do and report - 1

- **Compile and test your fetch_line program using a driver program, for example fetch_line_demo.c**
- **Report the output you get after running your driver program**

sample of expected output

```
$ ./fetch-line-demo < fgets_demo.c
trimmed line 3: int main(void)
trimmed line 4: {
trimmed line 5: char buf[BUFLEN];
* * * reading error: input line 6 too long for fetch_line's buf[40]
```

Things to do and report - 2

- **Create a sample input file with gradually increasing line lengths, for example like below**

```
35xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
36xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
37xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
---- more lines ---here
42xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

where the number at the beginning of each line equals the total length of that line.
What happens when you feed this file to your program?
Explain what you observe.

Things to do and report - 3

- **Add whitespaces (spaces, tabs, empty lines) and comments in various parts of the file you created in the last exercise. Verify that your program behaves as expected.**