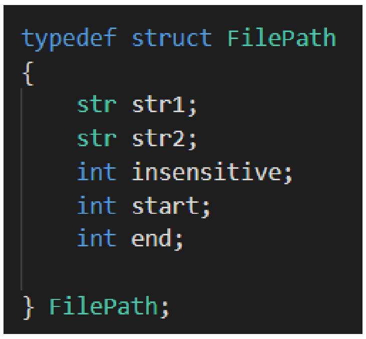
Homework 1 Report

CSE 344, Spring 2022

Sena Nur Ulukaya
1901042622

1. Design Decisions



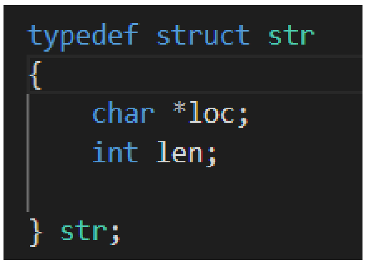
```c
typedef struct FilePath
{
    str str1;
    str str2;
    int insensitive;
    int start;
    int end;

} FilePath;
```

- FilePath struct represents “/str1/str2/”.

It has str1 and str2 fields and also indicators for insensitive(i), start(^), and($) cases.

- If insensitive == 1, inputFile address will be replaced with a copy of it with lower case characters and a lowercase path to compare without cases.

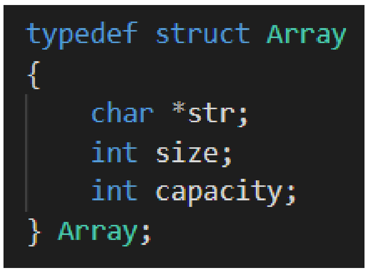
- If start == 1, it will only check the start of each line.
- If end == 1, it will only checks the end of each line.



```c
typedef struct str
{
    char *loc;
    int len;

} str;
```

- str struct represents a string in a FilePath.

loc field holds the address of the beginning of the string.
len field holds the length of the string.



```c
typedef struct Array
{
    char *str;
    int size;
    int capacity;
} Array;
```

- Array struct represents a data structure with size and capacity.
- I used it for OutputString. Because it is easier to reallocate and append.

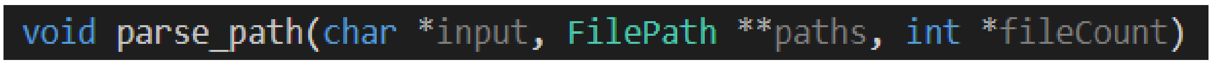
I have used FilePath struct to keep my paths and access its properties. I designed str struct with an address because I didn’t want to dynamically allocate a memory every time. With this design, I pointed to the beginning of the string, and with length property, I accessed the whole string.

2. Problem Solving Method

First of all, I checked the arguments. If they’re not valid for the format, the program will exit with an error message.

After confirming the argument count, I separated the file name and path string.

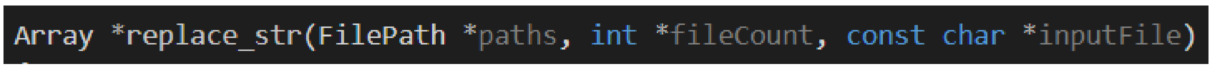
2.1 Parse Path



```
void parse_path(char *input, FilePath **paths, int *fileCount)
```

- This function separates /str1/str2/ into str1 and str2 and also takes the address of the strings and lengths.
- It also takes multiple paths by checking the ";" character and its afterward. It adds each new file path to the paths array of FilePaths.
- If the command is insensitive, it will make the insensitive field of the path "1".
- If the command has ^ at the beginning, it updates the location and length without ^ character and makes start property 1.
- If the command has $ at the beginning, it updates the location and length without $ character and makes end property 1.
- At the end of the function, each string will be separated with updated properties.

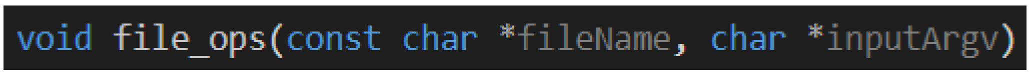
2.2 Replace



```
Array *replace_str(FilePath *paths, int *fileCount, const char *inputFile)
```

- It takes paths, path count, and also inputFile string and returns output String with replaced chars.
- It starts from the beginning of the input String and compares with each str1 of each command.
- For each command path, another function will check if str1 matches any location of the input String. (if there's no special case which indicates in design decisions)
- If there's "*" char, it will take the repeated char from the path and loops until repetition is over or continues if there are zero repeats from that char.
- Also, for the array of chars, if the next char of the input String matches any char from the array it will accept.
- If there are both array and * chars, it will accept any number of repetitions of chars from that array.
- At the end of the function output string will be filled with newly replaced chars.

2.3 Read/Write from/to File



```
void file_ops(const char *fileName, char *inputArgv)
```

- It takes the fileName and command file path.
- The file will be opened for reading and for writing I will create a temp file and rename it afterward. Because reading and writing can be not finished in one loop.
- After opening a file it will read the file string into a buffer and call the parsing and replacing functions I mentioned above.

- Then it writes the output string of replaced chars to a temp file until there's no byte left from reading.
- TempFile will be renamed with the first FileName from arguments and the old one will be removed.

2.4 Errors

- All the file operations used (open/read/write/close) are checked with error conditions and exited with an error message if there's one.
- There should be a char before * in the command to indicate which char will be repeated. It will exit and give an error.
- If argc != it will exit and give an error.
- If there are invalid chars in the path it will give an error and exit. (only letters, digits and *,/,^,$,[],;)

3. Requirements

a) `./hw1 '/str1/str2/' inputFilePath`
This will replace all occurrences of `str1` with `str2` in the file `inputFilePath` (relative or absolute)

b) `./hw1 '/str1/str2/i' inputFilePath`
This will perform the same as before, but will be case insensitive.

c) `./hw1 '/str1/str2/i;/str3/str4/' inputFilePath`
It must be able to support multiple replacement operations.

d) `./hw1 '/[zs]tr1/str2/' inputFilePath`
It must support multiple character matching; e.g. this will match both `ztr1` and `str1`

e) `./hw1 '/^str1/str2/' inputFilePath`
It must support matching at line starts; e.g. this will only match lines starting with `str1`

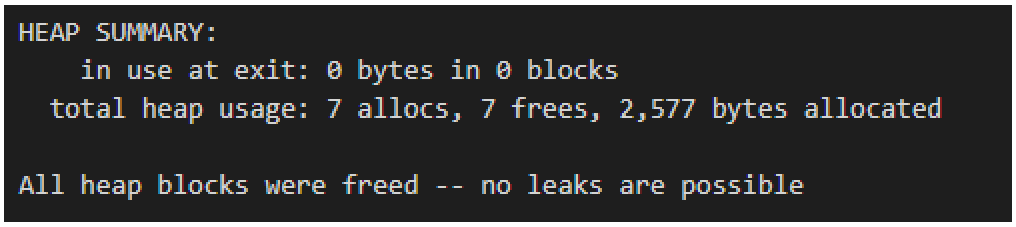
f) `./hw1 '/str1$/str2/' inputFilePath`
It must support matching at line ends; e.g. this will only match lines ending with `str1`

g) `./hw1 '/st*r1/str2/' inputFilePath`
It must support zero or more repetitions of characters; e.g. this will match `sr1`, `str1`, `sttr1`, `stttr1`, etc

And of course it must support arbitrary combinations of the above;
e.g. `./hw1 '/^Window[sz]*/Linux/i;/close[dD]$/open/' inputFilePath`

- It successfully accomplishes all the tasks above with the strategy and design above.
- The program won't let multiple instances with different replacement commands execute on the same file. If a program uses the file, another program can't write on that file. Because the program will lock the writing feature with the discussed method in the lecture.

- No memory leak checked with valgrind.



```
HEAP SUMMARY:
    in use at exit: 0 bytes in 0 blocks
  total heap usage: 7 allocs, 7 frees, 2,577 bytes allocated

All heap blocks were freed -- no leaks are possible
```