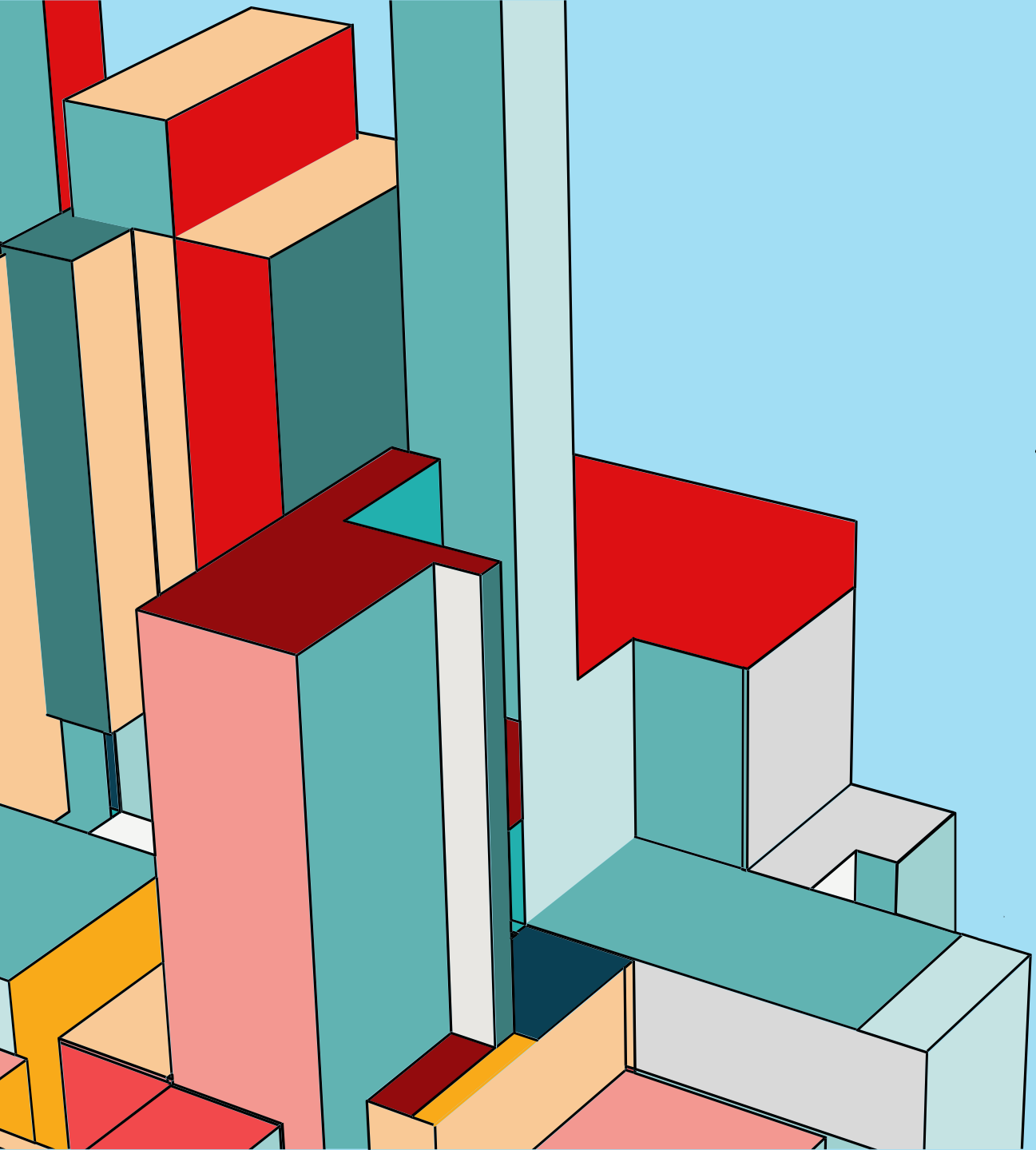
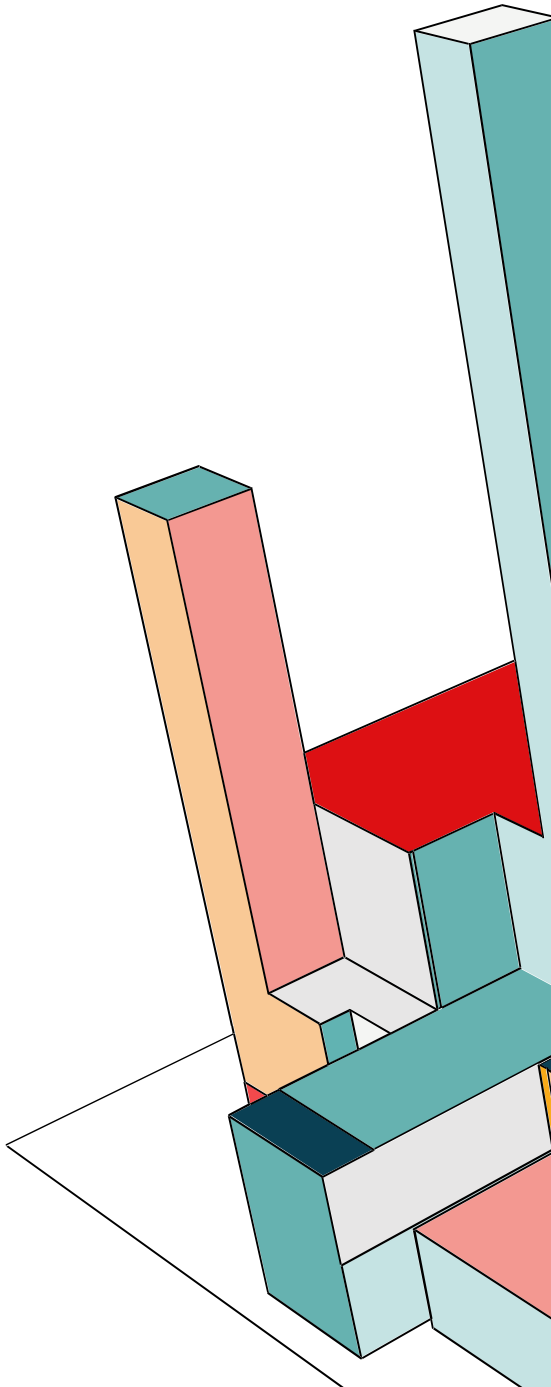


OPERATING SYSTEM PROJECT: CLOSH

Carlos Alberto Sánchez Calderón

FEATURES

- Built in commands
  - cd
  - md
  - rd
  - builtin
  - q
- External commands
- File redirection
- Background execution
- Errors
- Program design



```
>/home/so/Projects%cd
>/home%cd ./home/so/Projects
Error changing directory: No such file or directory
>/home%cd /home/so/Projects
>/home/so/Projects%cd ..
>/home/so%
```

CD

w/o arg goes to user home and w/ an argument to the indicated directory

```
>/home/so/Projects%ls

Basura  closh  closh.c  prueba
>/home/so/Projects%md fOld
>/home/so/Projects%ls

Basura  closh  closh.c  fOld  prueba
```

MD

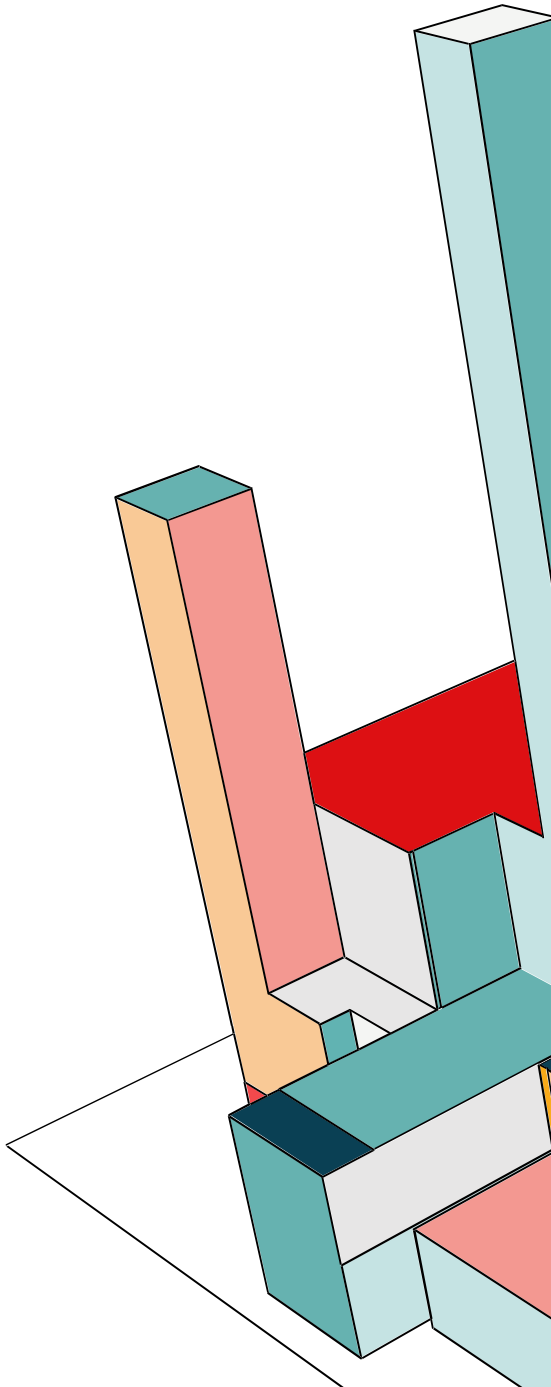
make a new directory named dir, dir can be a absolute path or relative path

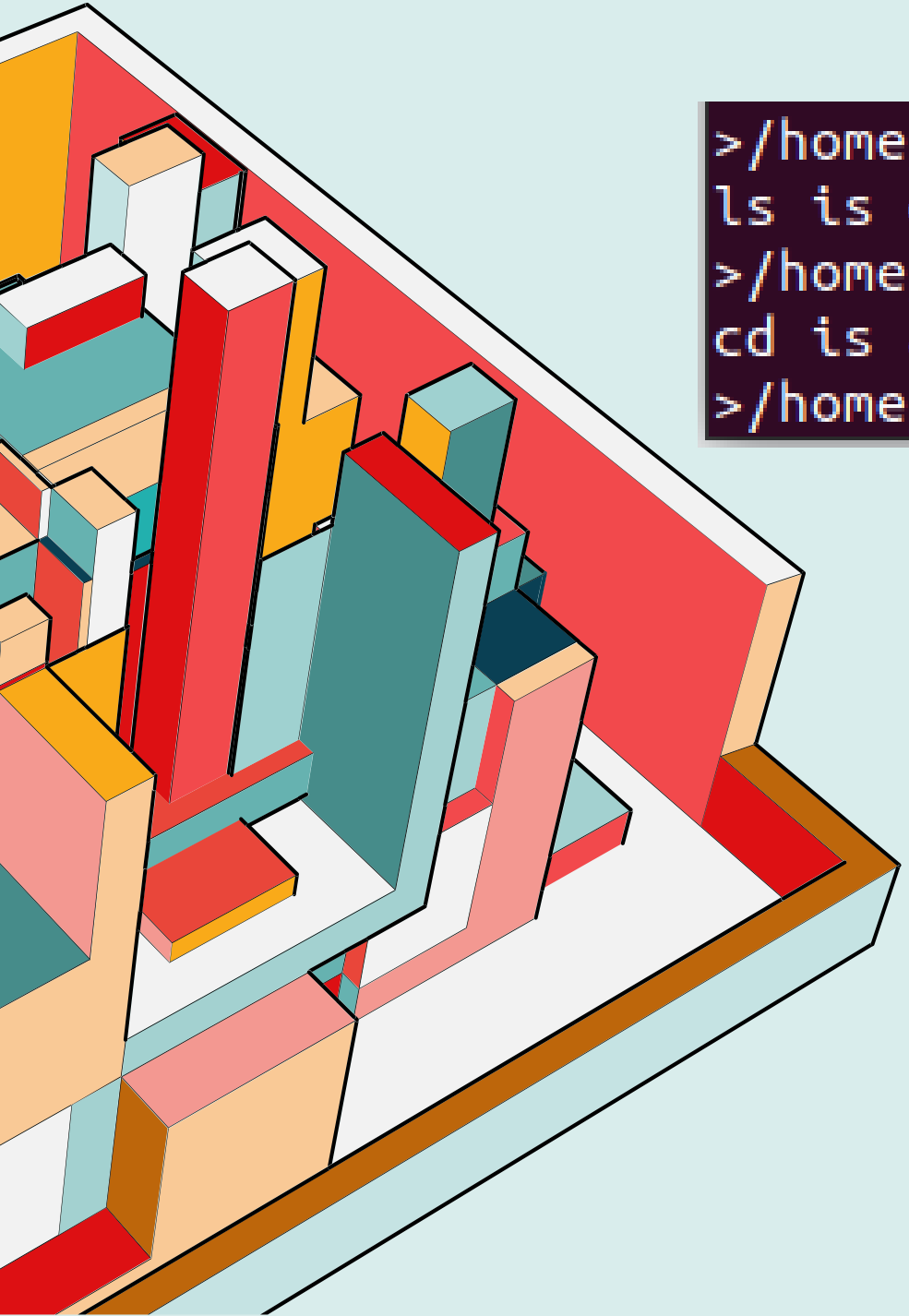
RD

remove the directory named dir

dir can be a absolute path or relative path

```
>/home/so/Projects%ls

Basura  closh  closh.c  fOld  prueba
>/home/so/Projects%rd fOld
>/home/so/Projects%ls

Basura  closh  closh.c  prueba
>/home/so/Projects%
```





```
>/home/so/Projects%builtin ls
ls is external
>/home/so/Projects%builtin cd
cd is a shell builtin
>/home/so/Projects%
```

BUILT IN

indicates if cmd is implemented in your shell or if it is external

```
>/home/so/Projects%./esp

hola
>/home/so/Projects%ls

Basura  closh  closh.c  esp  prueba
>/home/so/Projects%
```

EXTERNAL COMMANDS

All other commands are executed, including PATH env variables, use several arguments. The command is executed on a different process.

```
>/home/so/Projects%ls

Basura  closh  closh.c  esp  prueba
>/home/so/Projects%ls ~ output
>/home/so/Projects%ls = grep a

>/home/so/Projects%Basura
prueba
```

```
1
2 Basura
3 closh
4 closh.c
5 esp
6 output
7 prueba
```

FILE REDIRECTION

Implement the stdout redirection w/ the symbol "~"

Implement the pipe functionality w/ the symbol "="

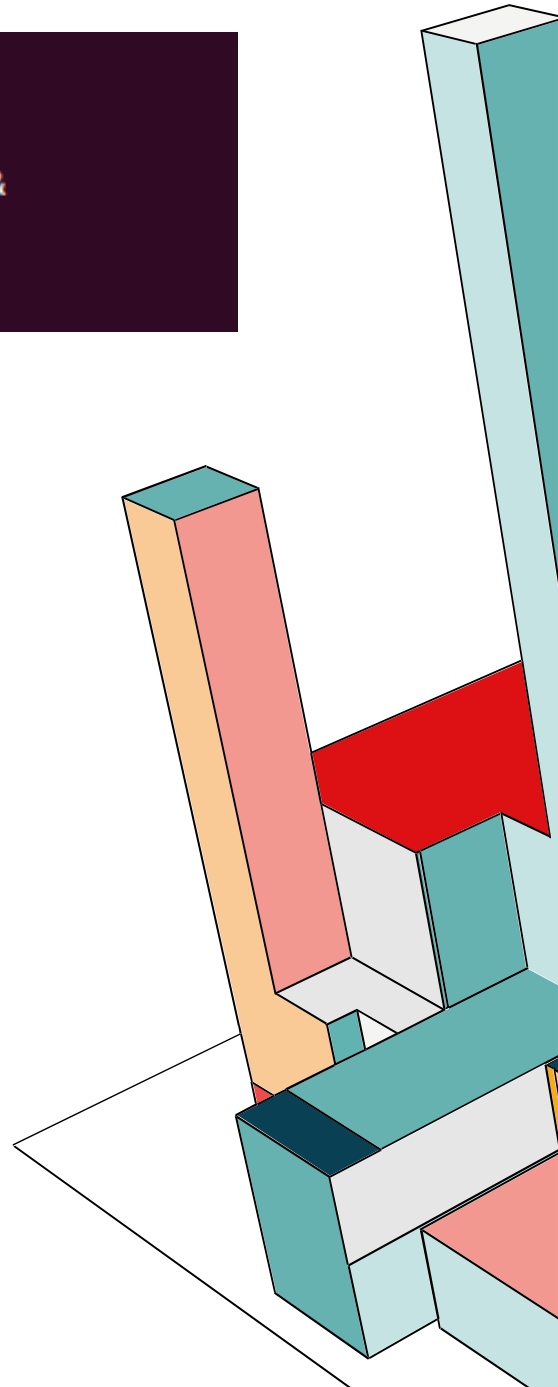
BACKGROUND EXECUTION AND ERRORS

```
>/home/so/Projects%./esp

hola
>/home/so/Projects%./esp &

>/home/so/Projects%hola
```

Typing an "&" at the end of a command should make it execute in the background

Send error messages to stderr

```
>/home/so/Projects%cd a
Error changing directory: No such file or directory
>/home/so/Projects%rd a
Error removing directory: No such file or directory
>/home/so/Projects%./eee

Error executing: No such file or directory
>/home/so/Projects%
```

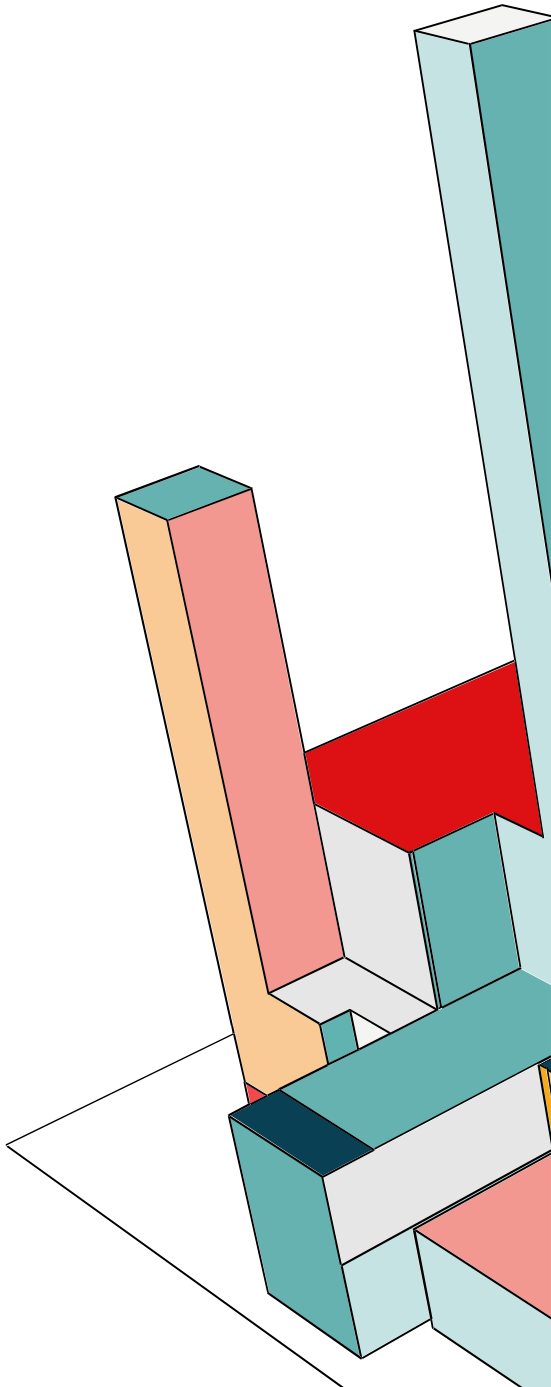


PROGRAM DESIGN

```c
/**
 * Simple shell interface program.
 *
 * Operating System Concepts - Tenth Edition
 * Copyright John Wiley & Sons - 2018
 *
 * Modified by Ilker Demirkol
 */
#include <stdio.h>···

#define MAX_LINE        80 /* 80 chars per line, per command */
#define READ_END    0
#define WRITE_END   1

/* Change directory */
void cd(char directory[30],int count){···

/* Create directory */
void md(char directory[30]){···

/* Delete directory */
void rd(char directory[30]){···

/* Print whether a cmd is builtin or not*/
void builtIn(char cmd[30]){···

/* Execute a external cmd */
void external_cmd(char entradas[11][50],int count,bool redirect){···

int main(void)···
```

Main functions of the program:

- Cd
- Md
- Rd
- Builtin
- External cmd
- Main



PROGRAM DESIGN: EXTERNAL CMD

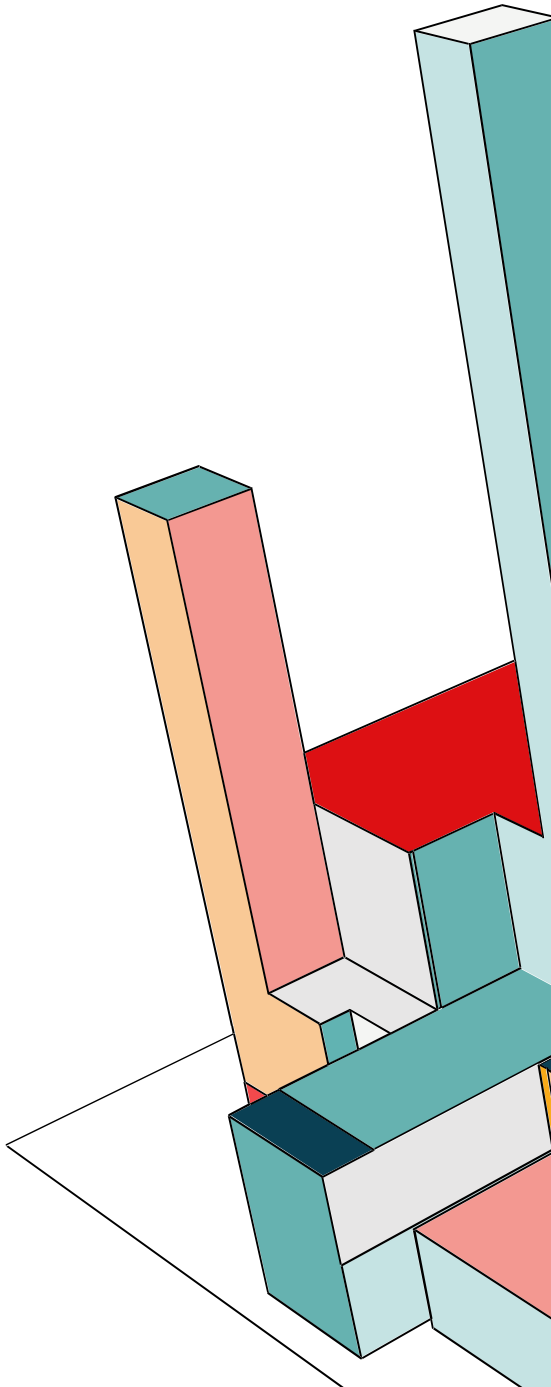
```c
/* Execute a external cmd */
void external_cmd(char entradas[11][50],int count,bool redirect){
    printf("\n");
    /* Create a new process*/
    pid_t pid;
    pid = fork();
    char *args[10 + 2];
    int args_count = 0;
    /* Modify the input char[][] to *char[] */
    while(args_count <= 12){
        if(args_count == count){
            break;
        }
        args[args_count] = malloc(101);
        strcpy(args[args_count],entradas[args_count]);
        args_count++;
    }
    if (redirect){
        args_count -= 2;
        args[args_count] = NULL;
    }else{
        args[args_count] = NULL;
    }
    if(pid < 0){
        fprintf(stderr,"Fork failed");
        return 1;
    }
    /* Child executes the cmd*/
    if(pid == 0){
        if(execvp(args[0], args) == -1){
            perror("Error executing");
        }
    }

    /* Parent decides if it waits for the child or not*/
    if(pid > 0){
        bool bg_mode = (entradas[count - 1][strlen(entradas[count - 1])-1]=='&');
        if(bg_mode == false)
        {
            wait(NULL);
        }
    }
    return 0;
}
```

Create a new process

The child process executes the command.

If the bg_mode is active the parent continue.

If not, the parent wait for the process to finish.



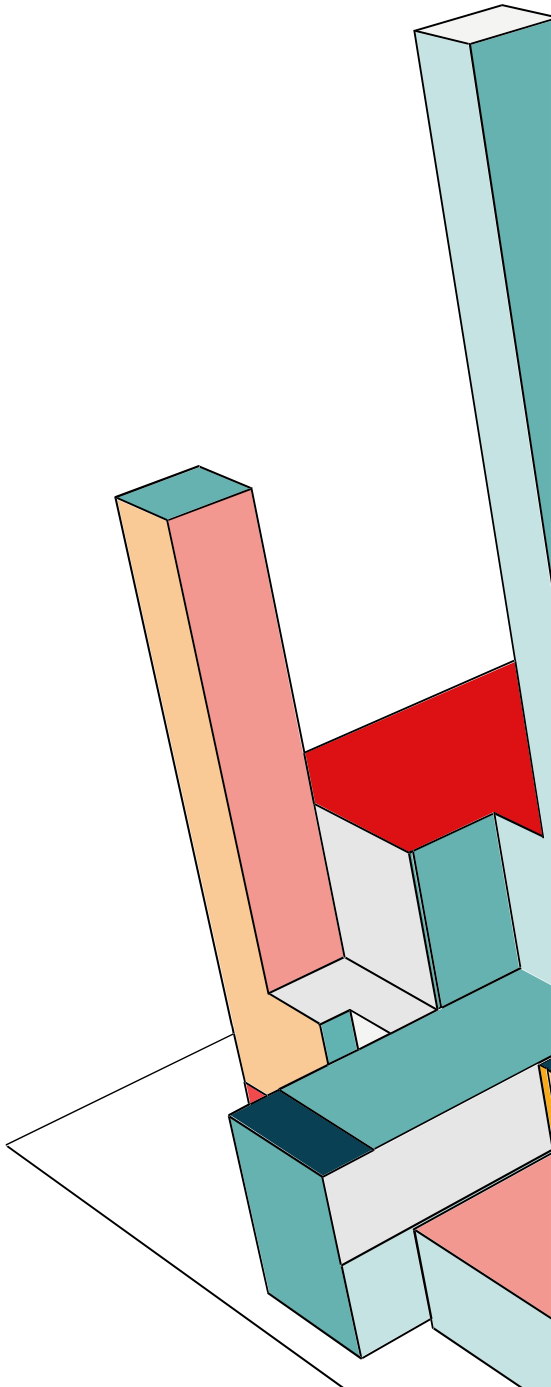
PROGRAM DESIGN: MAIN

```c
int main(void)
{
    char *args[MAX_LINE/2 + 1]; /* command line (of 80) has max of 40 arguments */
        int should_run = 1;
        while (should_run)
        {
            // get current directory
            char directory[30];
            getcwd(directory, sizeof(directory));
            printf(">");
            printf(directory);
            printf("%c",'%');

            /* Get user input */
            char entrada[100];
            char entradas[11][50];
            int count;
            FILE *fp;
            FILE *original_stdout_fd = dup(STDOUT_FILENO);
            FILE *original_stdin_fd = dup(STDIN_FILENO);
            fgets(entrada, sizeof(entrada), stdin);
            /* Look for the equal sign*/
            int pipe_position = -1;
            bool pipe_flag=false;
            for(int i = 0; i < 100; i++)
            {
                if(entrada[i] == '='){
                    pipe_position = i;
                }
            }
```

Print current directory

Gets user input

Search if the input has a pipe connection

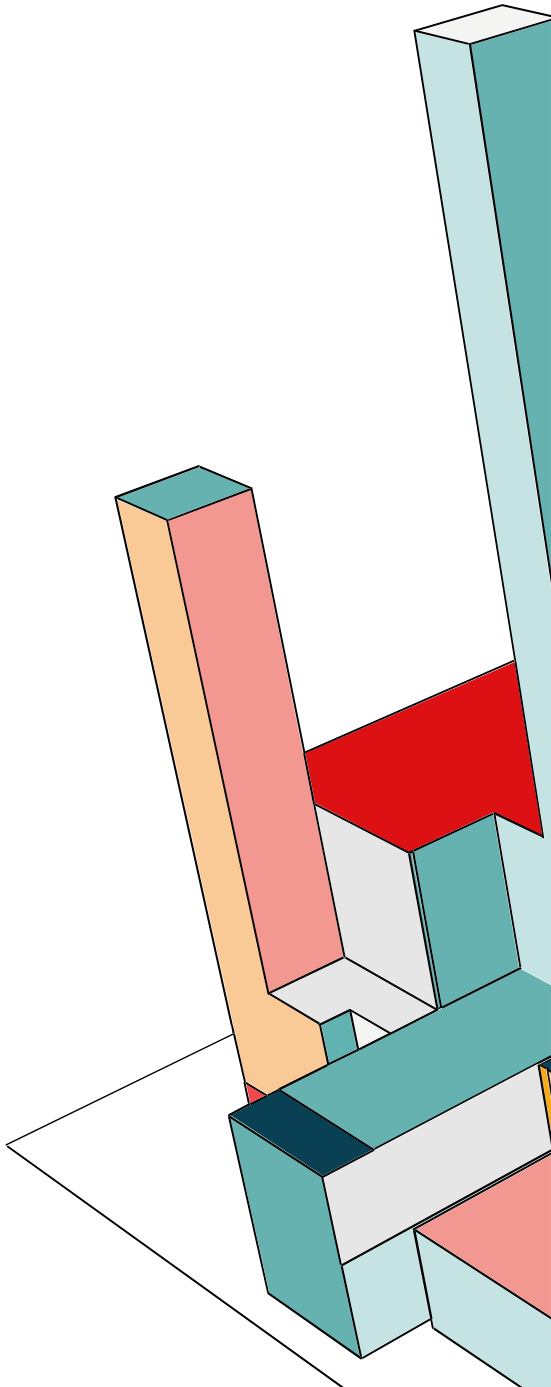


PROGRAM DESIGN: MAIN - PIPE

```c
pid_t pid;
int fd[2];
pipe(fd);
char msg[20];
/* Connect the output of one process to the input of the other */
if(pipe_position != -1)
{
    pid = fork();
    /* Parent process*/
    if (pid > 0)
    {
            /* Read the output */
        close(fd[WRITE_END]);
        read(fd[READ_END], msg, 20);
        int size_input = strlen(entrada) - 1;
        char ent[100];
        int cont_temp = 0;
        /* Set the entrada to the last part of the original input */
        for(int i = 0; i < size_input - pipe_position - 1;i++)
        {
            ent[i] = entrada[pipe_position + 1 + i];
            cont_temp++;
        }
        dup2(fd[READ_END], STDIN_FILENO);
        for(int i = 0; i < strlen(ent);i++)
        {
            entrada[i] = ent[i];
        }
        entrada[strlen(ent)] = '\0';
    }
    else
    {
        /* Change the stdout to the pipe */
        entrada[pipe_position] = '\0';
        close(fd[READ_END]);
        dup2(fd[WRITE_END], STDOUT_FILENO);
        pipe_flag = true;
    }
}
```

If there's a pipe it creates a process that will execute the first cmd and the result will be the input of the next cmd

- On the parent process it sets the stdInput to the pipe
- On the child process it sets the stdOut to the pipe

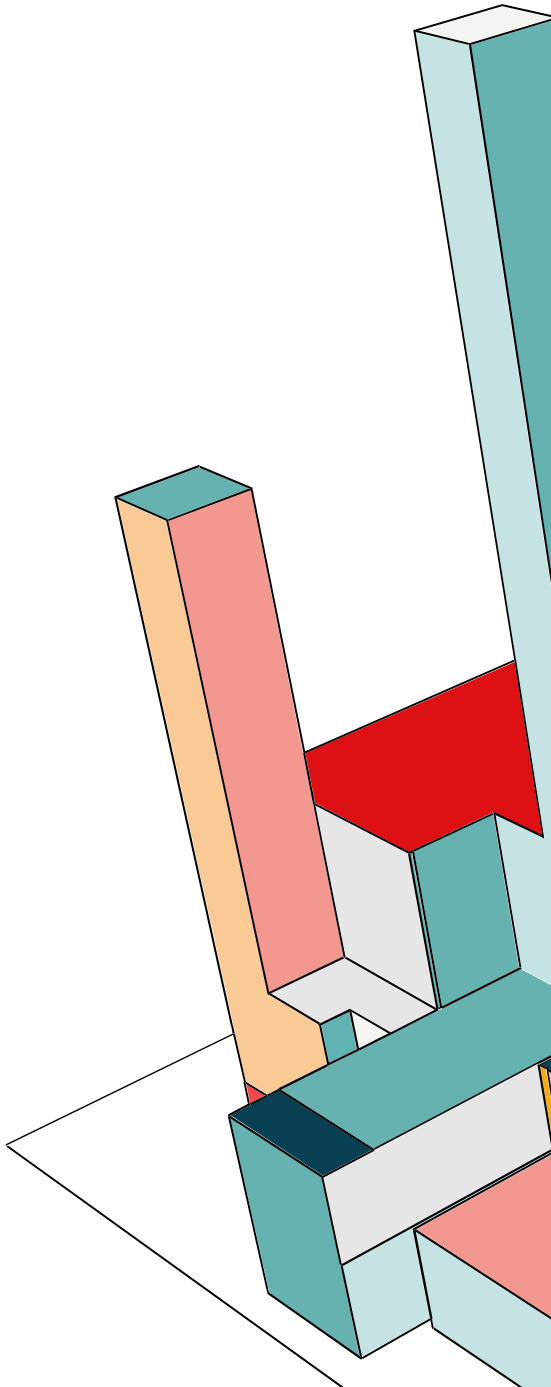


PROGRAM DESIGN: MAIN

```c
        /* Transform the entrada variable to an array */
        count = sscanf(entrada, "%s %s %s %s %s %s %s %s %s %s %s", &entradas[0], &entradas[1],
        bool flag = false;
        bool redirect = false;
        /* if there is ~ modify the stdout to the file*/
        if(strcmp(entradas[count - 2],"~") == 0)
        {
            redirect = true;
            fp = fopen(entradas[count - 1], "w");
            dup2(fileno(fp), STDOUT_FILENO);
        }
        /* if the first arg is cd calls to cd function */
        if(strcmp(entradas[0],"cd") == 0){ ...
        /* if the first arg is md calls to md function */
        if(strcmp(entradas[0],"md") == 0){  ...
        /* if the first arg is rd calls to rd function */
        if(strcmp(entradas[0],"rd") == 0){  ...
        /* if the first arg is builtin calls to builtin function */
        if(strcmp(entradas[0],"builtin") == 0){ ...
        /* if the first arg is q finish the program */
        if(strcmp(entradas[0],"q") == 0){   ...
        /* calls to the external function */
        if(flag == false){ ...
        /* Close all files */
        if(redirect){ ...
        if(pipe_flag){ ...
        close(fd[READ_END]);
        dup2(original_stdin_fd, STDIN_FILENO);
    }
    return 0;
}
```

Main calls for the appropriate function

Close all files that were open



CONCLUSION

During this class I have learned about different subjects of how a operating system works. With this project I had the opportunity to implement different features and put in practice the knowledge I acquired over the class.

One of the things that I used when developing this project is processes and how to create them. This helped me solidify my knowledge.

The feature that I had more difficult when implementing was the pipe functionality.

Overall, this projects was a very good way to improve my knowledge not only on my coding skills, but more important on how an operating system works under the hood

THANK YOU