

”

E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Laboratório de Programação

CÓDIGO: 21178

DOCENTE: Nelson Russo

A preencher pelo estudante

NOME: Leonardo Rosendo Silva

N.º DE ESTUDANTE: 2305494

CURSO: [2105] Licenciatura em Engenharia Informática

TRABALHO / RESOLUÇÃO:

O desenvolvimento do E-fólio B teve como uma das principais características a modularização e organização do código. Inicio este relatório ao apresentar a organização, a nível de ficheiros e módulos, contendo as seguintes pastas:

- Pasta inicial/base, contendo os executáveis *MAC_AF3* e *WIN_AF3.exe*, os ficheiros *main.c*, *aux.c*, as pastas *draw_rectangles*, *handle_commands*, *headers*, *unity_tests* e também um ficheiro *README.md*.
- A ***draw_rectangles***, contém os ficheiros *draw_rectangle.c* e *save_rectangle_coordinates.c*
- ***handle_commands***, contendo o ficheiro *command_reader.c*
- ***headers***, contendo os ficheiros *command.h*, *drawing.h* e *helper.h*.
- Para finalizar, a pasta ***unity_test***, contendo os ficheiros *creating_and_moving.c*, *gravity_collision_commands.c* e os respetivos executáveis para Windows e para MacBook.

Para exemplificar, segue uma imagem do VSCode:

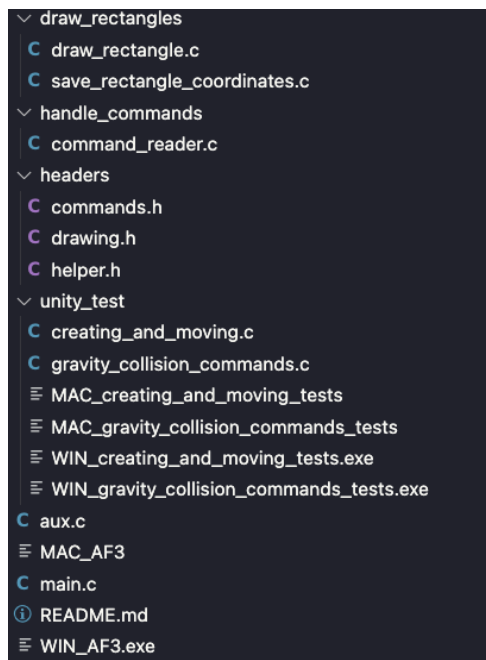


Imagem 1.1 - pastas e ficheiros

Passando agora para os detalhes e pontos importantes de cada ficheiro. Deixo uma observação de que o início da frase poderá conter uma letra minúscula por representar o nome do ficheiro.

draw_rectangle.c, neste ficheiro concentro as funções que irão alterar o retângulo de alguma forma, seja aplicar a “gravidade”, mover ou aplicar na matriz.

Este contém as funções, **Update_rect**, que irá aplicar a gravidade aos retângulos existentes, sendo esta uma função recursiva que irá “rodar” enquanto houver retângulos que não estão no “chão” ou acima de outros.

Update_matrix, que, com base na *linked list* **l_rect* irá preencher a matriz 80x25, os pontos da matriz podem receber -1, o ID do retângulo ou o ID do retângulo + 10.

Draw_matrix, após a matriz ser atualizada com os valores possíveis, esta será desenhada, caso o ponto observado seja -1, imprime-se “X”, se for entre 0 e 9 (possíveis ID’s) “+”, e se for mais que 9, imprime-se o valor -10, este valor indicará o ID do retângulo e estará sempre na casa mais acima e a esquerda do retângulo.

Valid_action, verifica se nas coordenadas indicadas existem um retângulo que não o passado nos parâmetros, usado neste caso para validar se a ação (criar ou mover) é válida.

Move_rectangle, com base nas coordenadas recebidas verifica se há um retângulo no ponto indicado e, caso exista, se a ação de mover para a direita ou esquerda é válida ou não. Em seguida, move o retângulo ou informa do erro.

Por fim, há também a função **Delete_rectangle**, que servirá para verificar se nas coordenadas recebidas existe um retângulo e, caso exista, apagá-lo. Caso contrário, aviso o usuário de que não existe.

Passando para o ficheiro *save_rectangle_coordinates.c*, temos as 3 funções que irão, de alguma forma, alterar os dados presente na *linked list* do tipo *LRect*, que será explicada mais a frente no relatório. Neste ficheiro temos as seguintes funções, **Add_rect**, **Free_rect** e **Free_all_rect**, que servirão para adicionar um retângulo à lista, libertar da memória apenas um item da lista e libertar todos os itens da lista respetivamente.

command_reader.c, neste temos todas as funções relacionadas à leitura de comandos, começando pela **Read_commands**, que irá solicitar ao utilizador o comando desejado, caso o seja “exit” ou “help” o programa irá fechar ou apresentar

os possíveis comandos existentes respetivamente.

Para validar o comando inserido, usa-se a função **Command_verify**, que por sua vez, irá “dividir” o input em partes, primeiro percebendo qual o comando (create, delete, moveright ou moveleft), em seguida pegamos a segunda parte do comando, as coordenadas, que podem ser “x,y+l,h”, “x,y+p” ou “x,y” a depender do comando. Neste ponto verifica-se se o número de argumentos, com base no comando, é válido com a função **Valid_amount_of_arguments**, sendo válido verifica-se quais são as coordenadas (x,y) com a função **Coordinates_verify**, caso tudo esteja correto com as coordenadas passamos para a terceira e última parte, que não se aplica para o comando “delete”, que irá verificar a segunda parte da coordenada passada, “l,h” ou “p”. Esta parte é tratada pela função **Verify_last_part**, cada verificação emite o seu devido aviso se houver algum erro, caso contrário os dados são devidamente passados para a *struct SCoordinates *s_coordinates*, que também será explicada mais abaixo.

Seguindo a ordem da imagem 1.1, agora temos a pasta *headers*, contendo os 3 ficheiros “.h”, *commands.h*, *drawing.h*, *helper.h*, armazenando as funções utilizadas em múltiplos ficheiros e a definição das estruturas de dados. Desta forma acredito não ser necessário mais enfoque nestes três ficheiros.

Neste ponto do relatório irei pular a pasta *unity_test* para dar a devida atenção em um tópico voltado para os testes em questão.

Passando agora para o ficheiro *aux.c* temos funções que irão auxiliar o programa em diferentes formas. Listando as funções existentes neste ficheiro temos

Str_to_lower, que irá permitir que o programa não seja *key sensitive*.

Get_str_input, função que solicita uma string ao usuário. **Initialize_matrix**, que inicia a matriz com o número -2. **Random_id**, que gera um número de identificação aleatório de 0 a 9. **Rectangle_len**, para verificar o tamanho da *linked list LRect*.

Print_commands, para imprimir a lista de todos os comandos.

Is_there_a_rectangle, retornado “EXIT_SUCESS” ou “EXIT_FAILURE” se houver ou não um retângulo nas coordenadas passadas, podendo ter 2 modos, recebendo apenas as coordenadas x e y ou x,y,h e l.

Para finalizar temos a função **Collision_warning**, que verifica que se o retângulo está no “chão” ou acima de outro retângulo e, se for o caso, significa que a

gravidade já terminou de exercer o seu “poder” neste, chamando assim a função **Collision_detectio**, que irá verificar se o retângulo observado colide com algum outro à direita ou esquerda e imprime o devido aviso quando o mesmo acontece. Neste aviso teremos a indicação do lado em que ocorre a colisão e os devidos números de identificação de cada um dos retângulos.

Antes de dar seguimento e falar do ficheiro *main.c*, vejamos as estruturas de dados utilizadas neste projeto. Ao todo foram criadas apenas 2 estruturas, que serviram perfeitamente para o seu propósito e eliminaram a necessidade de existirem mais. Temos a estrutura *SCoordinates*, cotendo 5 inteiros, *x*, *y*, *l*, *h* e *p*, e um char *command*, este poderá receber as letras c, l, r ou d, para *create*, *moveleft*, *moveright* e *delete*.

Para além desta temos uma *linked list* chamada *LRect*, que contém a estrutura *SRect*, esta lista servirá para armazenar todos os retângulos, contendo 5 inteiros, *x*, *y*, *l*, *h* e *id*, e claro a estrutura *SRect* para apontar o próximo nó da lista.

Para exemplificar de melhor forma, poderá observar a imagem 1.2 abaixo.

```
typedef struct
{
    char command; // can be 'c', 'l', 'r' or 'd' for create, move left, move right and delete
    int x;
    int y;
    int l; // width
    int h; // height
    int p; // number of positions to move.
} SCoordinates;

typedef struct SRect
{
    int x;
    int y;
    int l;
    int h;
    int id;
    struct SRect *next;
} LRect;
```

imagem 1.2, estrutura de dados

Optei por não utilizar a *SCoordinates* dentro da *SRect*, apesar de conter itens extremamente parecidos, para deixar as suas funcionalidades dentro do código bem separadas e facilitar a manutenção caso exista mais itens necessário para qualquer uma das duas a serem acrescentados em um futuro.

Menção ao ficheiro README.md, que contém os comandos para copiar e colar no terminal de modo a compilar o código, caso utilize um terminal baseado em UNIX.

Dentro do ficheiro *main.c* temos o funcionamento do programa, iniciando a lista *LRect *l_rect* e a estrutura *SCoordinates *s_coordinates*, em seguida entramos em um *while loop*, que irá se manter ativo enquanto a função **Read_commands** não receber um “exit” como comando.

Após receber o input do utilizador, inicializa-se a matriz sem os valores dos retângulos, apenas uma matriz vazia, em seguida verifica-se qual o comando recebido para executar a sua devida função, se é criar, mover ou deletar. Aqui, caso seja criar, confirma-se também se já existem 10 retângulos, se for o caso emite um aviso de que não foi possível criar e a razão para tal.

Após a execução do comando aplica-se a gravidade, atualiza-se os valores da matriz e a mesma é “desenhada” ou impressa no ecrã do utilizador. Por fim, quando saímos do *while loop*, liberta-se a memória utilizada para as respetivas variáveis.

Para uma melhor exemplificação, pode consultar a imagem 1.3, do ficheiro *main.c* (que também estará disponível no fim do relatório em texto).

Tendo agora uma visão geral dos arquivos e funções, com exceção à pasta *unity_test* que será explicada a seguir, falemos sobre como foi adaptar o código da AF3 para o E-fólio B.

Inicialmente os retângulos não eram preenchidos com o “+” e com o seu ID, gerando a necessidade de alterar esta parte, que foi extremamente simples e rápida, já que antes o interior era preenchido com o ID do retângulo (mas não era mostrado ao utilizador), além disso optei por colocar o ID na primeira casa disponível dentro do

```

int main()
{
    SCoordinates *s_coordinates;
    LRect *l_rect;
    int matrix[MAX_LINES][MAX_COL];

    l_rect = NULL;
    s_coordinates = (SCoordinates *)malloc(sizeof(SCoordinates));
    if (s_coordinates == NULL)
    {
        printf("Error, insufficient memory (coordinate)");
        return EXIT_FAILURE;
    }

    while (Read_commands(s_coordinates))
    {
        Initialize_matrix(matrix);

        if (s_coordinates->command == 'c')
        {
            if (Rectangle_len(l_rect) == 10)
                printf("\nLimit of 10 rectangles reached. To be able to create a new reactangle, please delete one (or more) rectangle(s)\n");
            else
            {
                if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
                {
                    l_rect = Add_rect(l_rect, s_coordinates);
                    Collision_warning(l_rect, l_rect);
                }
            }
        }
        else if (s_coordinates->command == 'd')
            l_rect = Delete_rectangle(l_rect, s_coordinates);
        else
        {
            Move_rectangle(l_rect, s_coordinates);
            Update_rect(l_rect);
            Update_matrix(matrix, l_rect);
            Draw_matrix(matrix);
        }
    }

    free(s_coordinates);
    s_coordinates = NULL;
    l_rect = Free_all_rect(l_rect);
}

```

Imagem 1.3 - Arquivo *main.c*

retângulo, assim como nos exemplos. Esta mudança acarretou também por me fazer decidir optar por um tamanho mínimo de 3x3, que desta forma permite a existência de um interior, caso contrário não seria possível manter as bordas como “X” e o interior contendo o ID do retângulo. Um retângulo 2x2 me obrigaria a colocar o ID na borda do mesmo, o que não condiz com a estética do projeto e facilmente poderia deixar confuso e desorganizado.

Em seguida acrescentei o comando *delete x,y*, que foi também muito simples, já que apenas tive de acrescentar um comando onde a verificação das coordenadas já existia, e como são feitas por partes, não afetou a estrutura da função **Command_verify**, para além desta mudança foi criada a função **Delete_rectangle**, explicada acima.

A colisão lateral, apesar de ser a parte mais desafiante do E-fólio B (já que a AF3 estava concluída) também não gerou grande esforço, apenas tive de reutilizar funções existentes no código. Para implementar esta funcionalidade verifiquei, com a ajuda da função **Valid_action**, se a ação de mover para a direita ou esquerda apenas uma casa era válida, ora, sendo válida então não existem retângulos ao lado, caso contrário sabemos que está a colidir com algum. Por isso, em caso de não ser possível mover para a esquerda, salvam-se as coordenadas como se

Um exemplo de aviso na imagem 1.4, onde existem 4 colisões após a criação de do retângulo 7.

```
Your command (or "exit" to leave and "help" for a user manual): create 6,1+5,6  
-> Warning, the rectangle 7 is now colliding with the 0 on the right  
-> Warning, the rectangle 7 is now colliding with the 9 on the right  
-> Warning, the rectangle 7 is now colliding with the 1 on the left  
-> Warning, the rectangle 7 is now colliding with the 2 on the left
```

A large rectangular area filled with a uniform grid of small black dots, representing a game field or map.

```
X X X X X X X X X X X X .  
X 1 + + X X 7 + + X X 0 + + X .  
X X X X X X + + + X X X X X .  
X X X X X X + + + X X X X X .  
X 2 + + X X + + + X X 9 + + X .  
X X X X X X X X X X X X X X .
```

Imagina-se agora que criemos outro retângulo sem colidir com nenhum deles, nenhum aviso seria dado.

Página 8 de 37

podem ser criados, se existe a possibilidade de criar fora dos limites do mundo, se é possível criar um retângulo onde já existe outro e por fim se é possível mover retângulos de maneiras que não deveriam ser possíveis, como quando há outro ao lado.

No segundo ficheiro, *gravity_collision_commands.c*, verifica-se a segunda parte do programa, igualmente importante, mas que o ideal é focar apenas depois do básica estar a funcionar corretamente. Neste caso temos os testes para a gravidade, testando se os retângulos realmente “caem” após criados, também o teste para empilhar retângulos onde mistura a gravidade e o limite de cada um, já que são criados 8 retângulos empilhados, porém com a coordenada de 1 casa acima do necessário, fazendo com que “caiam” pelo menos 1 casa. Em seguida testam-se diversos comandos errados, seja com coordenadas impossíveis, escrito errado, argumentos a mais, a menos, etc. Por fim, são realizados dois testes, o teste de colisão, para verificar se o número de colisões está correto e o teste sugerido pelo E-fólio B.

Cada um dos testes tem extrema importância na funcionalidade geral do código, de modo que foram escolhidos especialmente para testar desde o básico até a integração de conceitos do projeto. Nas imagens 1.5 e 1.6 apresento uma visão resumida do arquivo de testes (que estará disponível mais abaixo em texto).

```
printf("\n\n-----\n");
printf("\nStarting tests, here you will see the output from every function and after that, the result\n");
printf("\n\n-----\n");

test1 = Max_rectangles(s_coordinates, matrix);
test2 = Outside_boundaries(s_coordinates);
test3 = Rectangle_areas(s_coordinates, matrix);
test4 = Moving_rectangle(s_coordinates, matrix);

printf("\n\n-----\n");
printf("\n\tRESULTS\n");
printf("\n\n-----\n");
printf("\n1st test, maximum number of rectangles: %s\n", test1 == EXIT_SUCCESS ? "Success!" : "Failed.");
printf("\n2nd test, world boundaries: %s\n", test2 == EXIT_SUCCESS ? "Success!" : "Failed.");
printf("\n3rd test, rectangles over rectangles: %s\n", test3 == EXIT_SUCCESS ? "Success!" : "Failed.");
printf("\n4th test, moving rectangles: %s\n", test4 == EXIT_SUCCESS ? "Success!" : "Failed.");
```

Imagem 1.5 - creating_and_moving.c

```
printf("\n\n-----\n");
printf("\nStarting tests, here you will see the output from every function and, after that, the result\n");
printf("\n\n-----\n");

test1 = Gravity(s_coordinates, matrix);
test2 = Stacking_rectangles(s_coordinates, matrix);
test3 = Wrong_command(s_coordinates, matrix);
test4 = Collision_test(s_coordinates, matrix);
test5 = Efolio_B_Test(s_coordinates, matrix);

printf("\n\n-----\n");
printf("\n\tRESULTS\n");
printf("\n\n-----\n");
printf("\n1st test, checking if the gravity works: %s\n", test1 == EXIT_SUCCESS ? "Success!" : "Failed.");
printf("\n2nd Test, stacking rectangles: %s\n", test2 == EXIT_SUCCESS ? "Success!" : "Failed.");
printf("\n3rd test, wrong commands: %s\n", test3 == EXIT_SUCCESS ? "Success!" : "Failed.");
printf("\n4th test, collision: %s\n", test4 == EXIT_SUCCESS ? "Success!" : "Failed.");
printf("\n5th efolioB suggestion: %s\n", test5 == EXIT_SUCCESS ? "Success!" : "Failed.");
```

Imagem 1.6 - gravity_collision_commands.c

Finalizo assim o meu relatório com os códigos criados em formato de texto a seguir:

draw_rectangle.c:

```
#include "../headers/drawing.h"

// Apply "Gravity" to the rectangle
int Update_rect(LRect *_l_rect)
{
    LRect *aux;
    int update_counting = 0;

    aux = *_l_rect;
    while (aux)
    {
        while (aux->y > 1 && Valid_action(_l_rect, aux->x, aux->y - 1, aux->h, aux->l, aux->id) == EXIT_SUCCESS)
        {
            aux->y -= 1;
            Collision_warning(_l_rect, aux);
            update_counting++;
        }

        aux = aux->next;
    }

    // This will prevent error in gravity when deleting or moving rect that came later on the linked list
    // Without this recursion when the a rect falls all the previous rectangles (in the linked list) above it won't fall.
    if (update_counting > 0)
        return Update_rect(_l_rect);

    return update_counting;
}

/*
Insert the rect into the matrix, there's 3 possible values:
-1, to indicate there's a rect border, the rect ID and the rect ID + 10.

The Draw_matrix function will know what to do with these values
*/
void Update_matrix(int m[MAX_LINES][MAX_COL], LRect *_l_rect)
{
    LRect *aux;
    int i, c;

    aux = *_l_rect;
    while (aux)
    {
        for (i = 0; i < aux->l; i++)
        {
            for (c = 0; c < aux->h; c++)
            {
                // Rect borders
                if (i == 0 || aux->x + i == aux->x + aux->l - 1 || c == 0 || aux->y + c == aux->y + aux->h - 1)
                    m[aux->y + c - 1][aux->x + i - 1] = -1;
                else
                    m[aux->y + c - 1][aux->x + i - 1] = aux->id;
            }
        }

        /* Checking the fist slot, to insert the rect ID instead a "+ ", but only if we have space to do so.

```

We are looking for the slot $x+1, h-1$, but only if there's the `aux->id` value, not a `-1`. If there's a `-1` it means we don't have space, otherwise we'll increase the `in` value in 10.

```
*/
if (m[aux->y + aux->h - 3][aux->x] == aux->id)
m[aux->y + aux->h - 3][aux->x] = aux->id + 10;
}
}
aux = aux->next;
}
}

/*
The matrix contains 4 possible values:
-2 -> To indicate there's no rect there
-1 -> To indicate there's a rect border
A number between 0 and 9 -> To indicate the ID, we'll print "+" when finding that value
A number between 10 and 19 -> Basically the ID + 10, the idea is to fill only the first rect
space with that value, to indicate the Draw_matrix to print it's ID instead "X" or "." or "+ ".
*/
void Draw_matrix(int m[MAX_LINES][MAX_COL])
{
int l, c;

for (l = MAX_LINES - 1; l >= 0; l--)
{
printf("\n");
for (c = 0; c < MAX_COL; c++)
{
if (m[l][c] == -1)
printf("X ");
else if (m[l][c] >= 0 && m[l][c] <= 9)
printf("+ ");
else if (m[l][c] >= 10)
printf("%d ", m[l][c] - 10);
else
printf(". ");
}
}
}

// When creating id must be -10.
int Valid_action(LRect *_l_rect, int x, int y, int h, int l, int id)
{
LRect *aux = *_l_rect;
int valid;

while (aux)
{
if (id != aux->id)
{
valid = (x + l <= aux->x) || (x >= aux->x + aux->l) || (y + h <= aux->y) || (y >= aux->y + aux->h);

if (!valid)
{
if (id == -10)
printf("\nInvalid coordinates, there's a rectangle in the desired area\n");
return EXIT_FAILURE;
}
}
}
}
```

```

}
}
aux = aux->next;
}
return EXIT_SUCCESS;
}

void Move_rectangle(LRect *_l_rect, SCoordinates *_s_coordinates)
{
    LRect *aux;

    aux = *_l_rect;

    while (aux)
    {
        // Finding the rect that contains the provided position.
        if (Is_there_a_rectangle(aux, *_s_coordinates, 0) == EXIT_SUCCESS)
        {
            if (*_s_coordinates->command == 'r')
            {
                if (aux->x + aux->l + *_s_coordinates->p - 1 <= MAX_COL && Valid_action(_l_rect, aux->x + *_s_coordinates->p, aux->y, aux->h, aux->l, aux->id) == EXIT_SUCCESS)
                {
                    aux->x += *_s_coordinates->p;
                    Collision_warning(_l_rect, aux);
                }
            }
            else
            {
                printf("\nIt's not possible to move the rectangle %d, %d positions to the right\n", aux->id, *_s_coordinates->p);
            }
            else if (*_s_coordinates->command == 'l')
            {
                if (aux->x - *_s_coordinates->p > 0 && Valid_action(_l_rect, aux->x - *_s_coordinates->p, aux->y, aux->h, aux->l, aux->id) == EXIT_SUCCESS)
                {
                    aux->x -= *_s_coordinates->p;
                    Collision_warning(_l_rect, aux);
                }
            }
            else
            {
                printf("\nIt's not possible to move the rectangle %d, %d positions to the left\n", aux->id, *_s_coordinates->p);
            }
        }
        return;
    }

    aux = aux->next;
}

printf("\nRectangle not found\n");
}

LRect *Delete_rectangle(LRect *_l_rect, SCoordinates *_s_coordinates)
{
    LRect *aux;

    // Checking if we need to delete the first created rectangle
    if (Is_there_a_rectangle(_l_rect, *_s_coordinates, 0) == EXIT_SUCCESS)
    {
        *_l_rect = Free_rect(_l_rect);
    }
}

```

```

return L_rect;
}

// Checking the remaing positions
aux = L_rect;
while (aux->next)
{
    // Finding the rect that contains the provided position.
    if (Is_there_a_rectangle(aux->next, s_coordinates, 0) == EXIT_SUCCESS)
    {
        aux->next = Free_rect(aux->next);
        return L_rect;
    }

    aux = aux->next;
}
printf("\nRectangle not found\n");
return L_rect;
}

```

save_rectangle_coordinates.c:

```

#include "../headers/commands.h"

LRect *Add_rect(LRect *L_rect, SCoordinates *s_coordinates)
{
    LRect *new;

    new = (LRect *)malloc(sizeof(LRect));

    if (new == NULL)
    {
        printf("\nError memory allocation, rectangle\n");
        return NULL;
    }

    new->h = s_coordinates->h;
    new->l = s_coordinates->l;
    new->x = s_coordinates->x;
    new->y = s_coordinates->y;
    new->id = Random_id(L_rect);

    new->next = L_rect;
    return new;
}

LRect *Free_rect(LRect *L_rect)
{
    LRect *aux;

```

```

aux = L_rect->next;
free(L_rect);
L_rect = aux;

return L_rect;
}

LRect *Free_all_rect(LRect *L_rect)
{
while (L_rect)
L_rect = Free_rect(L_rect);

return L_rect;
}

```

command_reader.c:

```

#include "../headers/helper.h"
#include "../headers/commands.h"

int Coordinates_verify(char *coordinates, SCoordinates *s_coordinates);
int Verify_last_part(char *last_part, SCoordinates *s_coordinates);
int Valid_amount_of_arguments(char *str, char command);

int Read_commands(SCoordinates *s_coordinates)
{
char input[50];

printf("\n\nYour command (or \"exit\" to leave and \"help\" for a user manual): ");
Get_str_input(input);
Str_to_lower(input);

while (Command_verify(input, s_coordinates) == EXIT_FAILURE)
{
printf("\n\nYour command (or \"exit\" to leave and \"help\" for a user manual): ");
Get_str_input(input);
Str_to_lower(input);
}

if (strcmp("exit", input) == 0)
return 0;
else if (strcmp("help", input) == 0)
{
printf("\nList of commands:\n");
Print_commands();
return (Read_commands(s_coordinates));
}
return 1;
}

int Command_verify(char *input, SCoordinates *s_coordinates)
{
char *token, token_backup[20];

```

```

if (input == NULL)
return EXIT_FAILURE;

// Getting the first part of the command, can be exit, help, create, moveleft, moveright or delete
token = strtok(input, " ");

// First case, exit or help.
if (strcmp(token, "exit") == 0 || strcmp(token, "help") == 0)
return EXIT_SUCCESS;

// Verify command
if (strcmp(token, "create") != 0 && strcmp(token, "moveright") != 0 && strcmp(token, "moveleft") != 0 &&
strcmp(token, "delete") != 0)
{
printf("\nInvalid input, heres a list of all valid commands\n");
Print_commands();
return EXIT_FAILURE;
}

// Now we know that the command is right, let's see the instruction.
if (token[0] == 'c' || token[0] == 'd')
s_coordinates->command = token[0];
else
s_coordinates->command = token[4];

// The 2nd part of our command can be "x,y+l,h", "x,y+p" or just "x,y" (for the delete case)
token = strtok(NULL, " ");

if (token == NULL)
{
printf("\nInvalid, missing coordinates\n");
return EXIT_FAILURE;
}

strcpy(token_backup, token);
if (Valid_amount_of_arguments(token_backup, s_coordinates->command) == EXIT_FAILURE)
return EXIT_FAILURE;

// The token will change due the strtok on Coordinates_verify, erasing the 2nd part of it.
// The strtok modify permantly the string, and we still need the "full" command to verify it's last part.
strcpy(token_backup, token);

// Check if there's more arguments than necessary
if (token == NULL)
{
printf("\nCommand invalid, more arguments than necessary\n");
return EXIT_FAILURE;
}

// The create and move instructions are different, create has 2 coordinates plus width and height, and move has 2
coordinates and a position.
// but delete has no coordinates
if (Coordinates_verify(token, s_coordinates) == EXIT_FAILURE)
return EXIT_FAILURE;

```

```

// There's no need to verify the last part of command when deleting, because there is no "last part"
if (s_coordinates->command == 'd')
return EXIT_SUCCESS;
else
return Verify_last_part(token_backup, s_coordinates);
}

int Coordinates_verify(char *coordinates, SCordinates *s_coordinates)
{
char *x_and_y, *token;
int valid_coordinate = 0;

if (coordinates == NULL)
return EXIT_FAILURE;

// Get the first part of the coordinates x and y.
token = strtok(coordinates, "+");

// Get the x value
x_and_y = strtok(token, ",");

valid_coordinate = atoi(x_and_y);
if (valid_coordinate <= 0 || valid_coordinate > 80)
{
printf("\nInvalid \"x\" coordinate, notice that 1 <= x <= 80\n");
return EXIT_FAILURE;
}
s_coordinates->x = valid_coordinate;

// Get the y value
x_and_y = strtok(NULL, ",");
if (x_and_y == NULL)
{
printf("\nInvalid, missing \"y\" value\n");
return EXIT_FAILURE;
}

valid_coordinate = atoi(x_and_y);
if (valid_coordinate < 1 || valid_coordinate > 25)
{
printf("\nInvalid \"y\" coordinate, notice that 1 <= y <= 25\n");
return EXIT_FAILURE;
}
s_coordinates->y = valid_coordinate;

// Check if there's more commands than the necessary
x_and_y = strtok(NULL, ",");
if (x_and_y != NULL)
{
printf("\nInvalid coordinate format, more elements than the necessary\n");
return EXIT_FAILURE;
}

return EXIT_SUCCESS;
}

```



```

// By last part I mean the the "l,h" or "p"
int Verify_last_part(char *last_part, SCoordinates *s_coordinates)
{
    char *token;
    int valid_number;

    // getting the las part of the string x,y+l,h or x,y+p.
    token = strtok(last_part, "+");
    token = strtok(NULL, "+");

    if (token == NULL)
    {
        printf("\nInvalid, missing the %s\n", s_coordinates->command == 'c' ? "\"l,h\" values" : "\"p\" value");
        return EXIT_FAILURE;
    }

    if (s_coordinates->command != 'c')
    {
        valid_number = atoi(token);
        if (valid_number <= 0)
        {
            printf("\nInvalid, the \"p\" value while moving must be a number greater than 1\n");
            return EXIT_FAILURE;
        }
        s_coordinates->p = valid_number;
        s_coordinates->l = 0; //
        s_coordinates->h = 0;

        // token must contain only the p value, nothing else, so the its length must be 1 or 2. 1 if p < 10 and 2 if p is > 10;
        if ((valid_number < 10 && strlen(token) > 1) || (valid_number > 10 && strlen(token) > 2))
        {
            printf("\nInvalid, more arguments than the necessary. When creating please insert only \"x,y+l,h\", when moving, only \"x,y+p\"");
            return EXIT_FAILURE;
        }
    }
    else
    {
        // Get the value of l
        token = strtok(token, ",");

        // check if the sum of x and l - 1 is lower than 80, to respect the canvas boundary.
        valid_number = atoi(token);
        if (valid_number < 3)
        {
            printf("\nInvalid, the \"l\" value must be a number greater than 2\n");
            return EXIT_FAILURE;
        }
        else if ((valid_number + s_coordinates->x - 1) > 80)
        {
            printf("\nInvalid, rectangle is bigger than the allowed. Notice that the sum of x and l - 1 must be lower than 81\n");
            return EXIT_FAILURE;
        }
        else
            s_coordinates->l = valid_number;
    }
}

```

```

// Get the value of h
token = strtok(NULL, ",");

if (token == NULL)
{
    printf("\nInvalid, missing h value\n");
    return EXIT_FAILURE;
}

// check if the sum of y and h - 1 is lower than 25, to respect the canvas boundary.
valid_number = atoi(token);
if (valid_number < 3)
{
    printf("\nInvalid, the \"h\" value must be a number greater than 2\n");
    return EXIT_FAILURE;
}
else if ((valid_number + s_coordinates->y - 1) > 25)
{
    printf("\nInvalid, rectangle is bigger than the allowed. Notice that the sum of y and h - 1 must be lower than 26\n");
    return EXIT_FAILURE;
}
else
    s_coordinates->h = valid_number;

s_coordinates->p = 0;
}

// Check if there's any unnecessary extra command
token = strtok(NULL, ",");
if (token != NULL)
{
    printf("\nInvalid, more arguments than the necessary. When creating please insert only \"x,y+l,h\", when moving, only \"x,y+p\"\n");
    return EXIT_FAILURE;
}

return EXIT_SUCCESS;
}

// The delete command has only the coordinates, no + signal.
int Valid_amount_of_arguments(char *str, char command)
{
    int i = 0;

    str = strtok(str, "+");
    while (str)
    {
        str = strtok(NULL, "+");
        i++;
    }

    if ((command == 'd' && i == 1) || (command != 'd' && i == 2))
        return EXIT_SUCCESS;

    printf("\nInvalid number of \"+\" signals\n");
    return EXIT_FAILURE;
}

```

aux.c:

```
#include "headers/helper.h"
#include "headers/drawing.h"
#include <ctype.h>
#include <time.h>

void Str_to_lower(char *s)
{
    int i;

    for (i = 0; s[i]; i++)
        s[i] = tolower(s[i]);
}

// Get user input without the '\n' in the end
void Get_str_input(char *input)
{
    scanf("%s", input);

    if (input[strlen(input) - 1] == '\n')
        input[strlen(input) - 1] = '\0';
}

void Initialize_matrix(int m[MAX_LINES][MAX_COL])
{
    int l, c;

    for (l = 0; l < MAX_LINES; l++)
        for (c = 0; c < MAX_COL; c++)
            m[l][c] = -2;
}

// Generates a random ID between 0 and 9;
int Random_id(LRect *_rect)
{
    LRect *aux;
    int id;

    srand(time(NULL));
    id = rand() % 10;
    if (_rect == NULL)
        return id;

    // Verify if there's no other ID like that
    aux = _rect;
    while (aux)
    {
        if (aux->id == id)
        {
            id = rand() % 10;
        }
        aux = aux->next;
    }
}
```

```

// restart the verification
aux = L_rect;
}
else
aux = aux->next;
}

return id;
}

int Rectangle_len(LRect *L_rect)
{
LRect *aux = L_rect;
int size = 0;

while (aux)
{
aux = aux->next;
size++;
}

return size;
}

void Print_commands()
{
printf("(x,y >=1, x <= 80 and y <= 25)\n");
printf("-> create x,y+l,h - Creates a rectangle where (x,y) are the coordinates of the bottom-left corner, and (l,h) are
the length and height, respectively.\n");
printf("-> moveright x,y+p - Moves the rectangle located at coordinates (x,y) to the right by p positions.\n");
printf("-> moveleft x,y+p - Moves the rectangle containing the point (x,y) to the left by p positions.\n");
printf("-> delete x,y - Delete the rectangle containning the point (x,y).\n");
printf("-> exit\n-> help\n");
}

// mode 0: receive only x and y.
// mode 1: receive all the coordiantes x,y,l and h
int Is_there_a_rectangle(LRect *L_rect, SCoordinates *s_coordinates, int mode)
{

int dont_exist;

if (mode == 0)
{
if (s_coordinates->x >= L_rect->x && s_coordinates->x < L_rect->x + L_rect->l && s_coordinates->y >= L_rect->y &&
s_coordinates->y < L_rect->y + L_rect->h)
return EXIT_SUCCESS;
else
return EXIT_FAILURE;
}

dont_exist = (s_coordinates->x + s_coordinates->l <= L_rect->x) || (s_coordinates->x >= L_rect->x + L_rect->l) ||
(s_coordinates->y + s_coordinates->h <= L_rect->y) || (s_coordinates->y >= L_rect->y + L_rect->h);
if (!dont_exist)
return EXIT_SUCCESS;
}

```

```

else
return EXIT_FAILURE;
}

/* We have already the "Valid_action" function which checks if it's possible to move the rect
in a certain direction. We can reuse it here, if there's a collision it means the rectangle can't
go even left or right a single "step".
We're only counting side collisions.
*/
int Collision_detection(LRect *all_rectangles, LRect *current_rectangle)
{
LRect *aux;
SCoordinates *s_coordinates_1, *s_coordinates_2;
int counter = 0;

s_coordinates_1 = NULL;
s_coordinates_2 = NULL;

// Check if there's a rectangle on the right
if (Valid_action(all_rectangles, current_rectangle->x + 1, current_rectangle->y, current_rectangle->h, current_rectangle->l, current_rectangle->id) == EXIT_FAILURE)
{
s_coordinates_1 = (SCoordinates *)malloc(sizeof(SCoordinates));
if (s_coordinates_1 == NULL)
{
printf("Error, insufficient memory (collision_dection1)");
return counter;
}
s_coordinates_1->h = current_rectangle->h;
s_coordinates_1->l = current_rectangle->l;
s_coordinates_1->x = current_rectangle->x + 1;
s_coordinates_1->y = current_rectangle->y;
}

// Check if there's a rectangle on the left
if (Valid_action(all_rectangles, current_rectangle->x - 1, current_rectangle->y, current_rectangle->h, current_rectangle->l, current_rectangle->id) == EXIT_FAILURE)
{
s_coordinates_2 = (SCoordinates *)malloc(sizeof(SCoordinates));
if (s_coordinates_2 == NULL)
{
printf("Error, insufficient memory (collision_dection2)");
return counter;
}
s_coordinates_2->h = current_rectangle->h;
s_coordinates_2->l = current_rectangle->l;
s_coordinates_2->x = current_rectangle->x - 1;
s_coordinates_2->y = current_rectangle->y;
}

aux = all_rectangles;

while (aux)
{
if (aux->id != current_rectangle->id)
{
if (s_coordinates_1 != NULL && Is_there_a_rectangle(aux, s_coordinates_1, 1) == EXIT_SUCCESS)

```

```

{
printf("-> Warning, the rectangle %d is now colliding with the %d on the right\n", current_rectangle->id, aux->id);
counter++;
}
if (s_coordinates_2 != NULL && Is_there_a_rectangle(aux, s_coordinates_2, 1) == EXIT_SUCCESS)
{
printf("-> Warning, the rectangle %d is now colliding with the %d on the left\n", current_rectangle->id, aux->id);
counter++;
}
}
aux = aux->next;
}

if (s_coordinates_1)
free(s_coordinates_1);
if (s_coordinates_2)
free(s_coordinates_2);

return counter;
}

// We don't want the collision detection system to be called if the "gravity" can act on it,
// in that case, we check if the rect is already on the bottom or on the top of another.
int Collision_warning(LRect *all_rect, LRect *current_rect)
{
SCoordinates *s_coordinates;
int counter = 0;

s_coordinates = (SCoordinates *)malloc(sizeof(SCoordinates));
if (s_coordinates == NULL)
{
printf("Error, insufficient memory (coordinate collision_warning)");
return counter;
}

s_coordinates->h = current_rect->h;
s_coordinates->l = current_rect->l;
s_coordinates->x = current_rect->x;
s_coordinates->y = current_rect->y - 1;

if (current_rect->y == 1 || Valid_action(all_rect, current_rect->x, current_rect->y - 1, current_rect->h, current_rect->l,
current_rect->id) == EXIT_FAILURE)
counter = Collision_detection(all_rect, current_rect);

free(s_coordinates);
return counter;
}

```

main.c:

```

#include "headers/helper.h"
#include "headers/drawing.h"
#include "headers/commands.h"

int main()
{

```

```

SCoordinates *s_coordinates;
LRect *l_rect;
int matrix[MAX_LINES][MAX_COL];

l_rect = NULL;
s_coordinates = (SCoordinates *)malloc(sizeof(SCoordinates));
if (s_coordinates == NULL)
{
    printf("Error, insufficient memory (coordinate)");
    return EXIT_FAILURE;
}

while (Read_commands(s_coordinates))
{
    Initialize_matrix(matrix);

    if (s_coordinates->command == 'c')
    {
        if (Rectangle_len(l_rect) == 10)
            printf("\nLimit of 10 rectangles reached. To be able to create a new reactangle, please delete one (or more) rectangle(s)\n");
        else
        {
            if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
            {
                l_rect = Add_rect(l_rect, s_coordinates);
                Collision_warning(l_rect, l_rect);
            }
        }
    }
    else if (s_coordinates->command == 'd')
        l_rect = Delete_rectangle(l_rect, s_coordinates);
    else
    {
        Move_rectangle(l_rect, s_coordinates);
        Update_rect(l_rect);
        Update_matrix(matrix, l_rect);
        Draw_matrix(matrix);
    }

    free(s_coordinates);
    s_coordinates = NULL;
    l_rect = Free_all_rect(l_rect);
}

```

commands.h:

```

#ifndef COMMANDS_H
#define COMMANDS_H

#include "helper.h"
#include "drawing.h"

int Read_commands(SCoordinates *s_coordinates);
int Random_id(LRect *l_rect);
LRect *Add_rect(LRect *l_rect, SCoordinates *s_condinates);
void Move_rectangle(LRect *l_rect, SCoordinates *s_coordinates);
int Rectangle_len(LRect *l_rect);
int Command_verify(char *input, SCoordinates *s_coordinates);

```

```
#endif
```

drawing.h:

```
#ifndef DRAWING_H
#define DRAWING_H
#include "helper.h"

void Update_matrix(int m[MAX_LINES][MAX_COL], LRect *l_rect);
void Draw_matrix(int m[MAX_LINES][MAX_COL]);
int Update_rect(LRect *l_rect);
int Valid_action(LRect *l_rect, int x, int y, int h, int l, int id);
LRect *Free_all_rect(LRect *l_rect);
LRect *Free_rect(LRect *l_rect);
LRect *Delete_rectangle(LRect *l_rect, SCoordinates *s_coordinates);

#endif
```

helper.h:

```
#ifndef HELPER_H
#define HELPER_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LINES 25
#define MAX_COL 80
typedef struct
{
    char command; // can be 'c', 'l', 'r' or 'd' for create, move left, move right and delete
    int x;
    int y;
    int l; // width
    int h; // height
    int p; // number of positions to move.
} SCoordinates;

typedef struct SRect
{
    int x;
    int y;
    int l;
    int h;
    int id;
    struct SRect *next;
} LRect;

void Str_to_lower(char *s);
void Get_str_input(char *input);
void Initialize_matrix(int m[MAX_LINES][MAX_COL]);
void Print_commands();
int Collision_detection(LRect *all_rectangles, LRect *current_rectangle);
int Is_there_a_rectangle(LRect *l_rect, SCoordinates *s_coordinates, int mode);
int Collision_warning(LRect *all_rect, LRect *current_rect);
```



```
#endif
```

creating_and_moving.c:

```
/*
Here we'll test if the code will allow the user to:
create only 10 rectangles;
create beyond the "world" boundaries
create a rect where there's another rectangle
move in way that is not supposed
*/

#include "../headers/helper.h"
#include "../headers/drawing.h"
#include "../headers/commands.h"

int Max_rectangles(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int Outside_boundaries(SCoordinates *s_coordinates);
int Rectangle_areas(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int Moving_rectangle(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int List_size(LRect *l_rect);

int main()
{
    SCoordinates *s_coordinates;
    int matrix[MAX_LINES][MAX_COL];
    int test1, test2, test3, test4;

    s_coordinates = (SCoordinates *)malloc(sizeof(SCoordinates));
    if (s_coordinates == NULL)
    {
        printf("Error, insuficient memory (coordinate)");
        return EXIT_FAILURE;
    }

    printf("\n-----\n");
    printf("\nStarting tests, here you will se the output from every function and after that, the result\n");
    printf("\n-----\n");
    test1 = Max_rectangles(s_coordinates, matrix);
    test2 = Outside_boundaries(s_coordinates);
    test3 = Rectangle_areas(s_coordinates, matrix);
    test4 = Moving_rectangle(s_coordinates, matrix);

    printf("\n\n-----\n");
    printf("\n\tRESULTS\n");
    printf("\n-----\n");
    printf("\n1st test, maximum number of rectangles: %s\n", test1 == EXIT_SUCCESS ? "Sucess!" : "Failed.");
    printf("\n2nd test, world boundaries: %s\n", test2 == EXIT_SUCCESS ? "Sucess!" : "Failed.");
    printf("\n3rd test, rectangles over rectangles: %s\n", test3 == EXIT_SUCCESS ? "Sucess!" : "Failed.");
    printf("\n4th test, moving rectangles: %s\n", test4 == EXIT_SUCCESS ? "Sucess!" : "Failed.");

    free(s_coordinates);
    s_coordinates = NULL;
}
```

```

// First test, the maximum amount of rectangles
int Max_rectangles(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
    LRect *l_rect;
    int i;

    l_rect = NULL;

    printf("\n1st Test, checking how many rectangles can the user create\n");
    Initialize_matrix(matrix);

    for (i = 1; i < 15; i++)
    {
        s_coordinates->h = i + 3;
        s_coordinates->l = 3;
        s_coordinates->x = i * 3;
        s_coordinates->y = 1;
        if (Rectangle_len(l_rect) == 10)
            printf("\nLimit of 10 rectangles reached. To be able to create a new reactangle, please delete one (or more)
            rectangle(s)\n");
        else
        {
            if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
                l_rect = Add_rect(l_rect, s_coordinates);
        }
    }

    Update_matrix(matrix, l_rect);
    Draw_matrix(matrix);

    if (List_size(l_rect) != 10)
    {
        l_rect = Free_all_rect(l_rect);
        return EXIT_FAILURE;
    }

    l_rect = Free_all_rect(l_rect);
    return EXIT_SUCCESS;
}

int List_size(LRect *l_rect)
{
    LRect *aux;
    int size = 0;

    aux = l_rect;

    while (aux)
    {
        size++;
        aux = aux->next;
    }

    return size;
}

```

```

int Outside_boundaries(SCoordinates *s_coordinates)
{
int good_to_go = EXIT_SUCCESS;
char *input;

printf("\n\n2nd Test, checking the \"world\" limits and sizes\n");
input = (char *)malloc(sizeof(char) * 18);
if (input == NULL)
{
printf("\nError while allocating memory (2nd test creating and moving)\n");
return EXIT_FAILURE;
}

// A x lower than 1
printf("\nx is lower than 1:");
strcpy(input, "create -1,1+3,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError, creating a rectangle with the x lower than 1\n");
good_to_go = EXIT_FAILURE;
}

// A x greater than 80
printf("\nx is greater than 80:");
strcpy(input, "create 81,1+3,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError, creating a rectangle with the x greater than 80\n");
good_to_go = EXIT_FAILURE;
}

// A y lower than 1
printf("\ny is lower than 1:");
strcpy(input, "create 1,-1+3,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError, creating a rectangle with the y lower than 1\n");
good_to_go = EXIT_FAILURE;
}

// A y greater than 25
printf("\ny is greater than 25:");
strcpy(input, "create 1,26+3,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError, creating a rectangle with the y greater than 25\n");
good_to_go = EXIT_FAILURE;
}

// Incoret height and length, should be greater than 2
printf("\nIncoret height and length, should be greater than 2:");
strcpy(input, "create 1,1+1,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError, creating a rectangle with an invalid length\n");
good_to_go = EXIT_FAILURE;
}
}

```

```

strcpy(input, "create 1,1+3,1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
    printf("\nError, creating a rectangle with an invalid height\n");
    good_to_go = EXIT_FAILURE;
}

// Checking if the user can create a rectangle with valid x and y values but h, l or h and l invalid
printf("\nValid x and y values but h, l or h and l invalid(s):");
strcpy(input, "create 76,1+6,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
    printf("\nError, creating a rectangle with an invalid size, invalid length\n");
    good_to_go = EXIT_FAILURE;
}

strcpy(input, "create 1,22+5,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
    printf("\nError, creating a rectangle with an invalid size, invalid height\n");
    good_to_go = EXIT_FAILURE;
}

strcpy(input, "create 76,22+6,5");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
    printf("\nError, creating a rectangle with an invalid size, invalid height and length\n");
    good_to_go = EXIT_FAILURE;
}

free(input);
return good_to_go;
}

int Rectangle_areas(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
    LRect *l_rect;
    int good_to_go = EXIT_SUCCESS;

    l_rect = NULL;
    printf("\n3rd Test, checking if the user can create a rectangle over another rectangle\n");
    Initialize_matrix(matrix);

    // Creating the first rectangle
    s_coordinates->h = 5;
    s_coordinates->l = 12;
    s_coordinates->x = 1;
    s_coordinates->y = 1;
    if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
    l_rect = Add_rect(l_rect, s_coordinates);

    // Trying the first invalid command
    s_coordinates->h = 5;
    s_coordinates->l = 12;
    s_coordinates->x = 8;

```

```

s_coordinates->y = 3;
if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
{
l_rect = Add_rect(l_rect, s_coordinates);
printf("\nError creating an rectangle over another, test 1\n");
good_to_go = EXIT_FAILURE;
}

// Trying the second invalid command, checking the limits
s_coordinates->h = 5;
s_coordinates->l = 12;
s_coordinates->x = 12;
s_coordinates->y = 5;
if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
{
l_rect = Add_rect(l_rect, s_coordinates);
printf("\nError creating an rectangle over another, test 1\n");
good_to_go = EXIT_FAILURE;
}

if (List_size(l_rect) > 1)
good_to_go = EXIT_FAILURE;

l_rect = Free_all_rect(l_rect);
return good_to_go;
}

int Moving_rectangle(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
LRect *l_rect, *aux;
int good_to_go = EXIT_SUCCESS;

l_rect = NULL;
printf("\n4th Test, checking if the user can move to wrong directions\n");
Initialize_matrix(matrix);

// Creating two rectangles next to each other
s_coordinates->h = 3;
s_coordinates->l = 3;
s_coordinates->x = 1;
s_coordinates->y = 1;
if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
l_rect = Add_rect(l_rect, s_coordinates);

s_coordinates->h = 5;
s_coordinates->l = 3;
s_coordinates->x = 4;
s_coordinates->y = 1;
if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
l_rect = Add_rect(l_rect, s_coordinates);

// Trying to move the 1st rect to the right
s_coordinates->x = 1;
s_coordinates->y = 1;
s_coordinates->p = 1;
s_coordinates->command = 'r';

```

```

Move_rectangle(l_rect, s_coordinates);

// Trying to move the 1st rect to the left
s_coordinates->x = 1;
s_coordinates->y = 1;
s_coordinates->p = 1;
s_coordinates->command = 'l';
Move_rectangle(l_rect, s_coordinates);

// Trying to move the 1st rect to the left
s_coordinates->x = 4;
s_coordinates->y = 1;
s_coordinates->p = 1;
s_coordinates->command = 'l';
Move_rectangle(l_rect, s_coordinates);

// Trying to move the 2nd rect to the right, should work
s_coordinates->x = 4;
s_coordinates->y = 1;
s_coordinates->p = 2;
s_coordinates->command = 'r';
Move_rectangle(l_rect, s_coordinates);

aux = l_rect;

// 2nd rectangle
if (aux->x != 6)
{
    printf("\nError while moving the 2nd rectangle\n");
    good_to_go = EXIT_FAILURE;
}

aux = aux->next;
// 1st rectangle
if (aux->x != 1)
{
    printf("\nError while moving the 1st rectangle, it wasn't supposed to move\n");
    good_to_go = EXIT_FAILURE;
}

return good_to_go;
}

```

gravity_collision_commands.c:

```

/*
Here we'll test:
The gravity system;
If a rectangle stays in the top of another rectangle
Invalid commands;
Collision;
*/

#include "../headers/helper.h"
#include "../headers/drawing.h"

```

```

#include "../headers/commands.h"
#include <time.h>

int Gravity(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int Stacking_rectangles(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int Wrong_command(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int Collision_test(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int Efolio_B_Test(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL]);
int main()
{
    SCoordinates *s_coordinates;
    int matrix[MAX_LINES][MAX_COL];
    int test1, test2, test3, test4, test5;

    s_coordinates = (SCoordinates *)malloc(sizeof(SCoordinates));
    if (s_coordinates == NULL)
    {
        printf("Error, insuficient memory (coordinate)");
        return EXIT_FAILURE;
    }

    printf("\n-----\n");
    printf("\nStarting tests, here you will see the output from every function and, after that, the result\n");
    printf("\n-----\n");

    test1 = Gravity(s_coordinates, matrix);
    test2 = Stacking_rectangles(s_coordinates, matrix);
    test3 = Wrong_command(s_coordinates, matrix);
    test4 = Collision_test(s_coordinates, matrix);
    test5 = Efolio_B_Test(s_coordinates, matrix);

    printf("\n\n-----\n");
    printf("\n\tRESULTS\n");
    printf("\n-----\n");
    printf("\n1st test, checking if the gravity works: %s\n", test1 == EXIT_SUCCESS ? "Sucess!" : "Failed.");
    printf("\n2nd Test, stacking rectangles: %s\n", test2 == EXIT_SUCCESS ? "Sucess!" : "Failed.");
    printf("\n3rd test, wrong commands: %s\n", test3 == EXIT_SUCCESS ? "Sucess!" : "Failed.");
    printf("\n4th test, collision: %s\n", test4 == EXIT_SUCCESS ? "Sucess!" : "Failed.");
    printf("\n5th efolioB suggestion: %s\n", test5 == EXIT_SUCCESS ? "Sucess!" : "Failed.");

    free(s_coordinates);
    s_coordinates = NULL;
}

// Here we'll create 10 rectangles, they will all have y > 1 but they wont be at top of another rectangle,
// in the end, they should be at the floor (y == 1).
int Gravity(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
    LRect *l_rect, *aux;
    int i, random_number, good_to_go = EXIT_SUCCESS;

    l_rect = NULL;

    printf("\n1st Test, checking if the gravity works\n");
    Initialize_matrix(matrix);
    srand(time(NULL));

```

```

for (i = 1; i <= 10; i++)
{
    s_coordinates->h = 5;
    s_coordinates->l = 3;
    s_coordinates->x = i * 4;

    random_number = (rand() % 10) + 2;
    s_coordinates->y = random_number;

    if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
        l_rect = Add_rect(l_rect, s_coordinates);
}

printf("\nFor testing purposes only, here's the matrix without gravity:\n");
Update_matrix(matrix, l_rect);
Draw_matrix(matrix);

printf("\n\nThe result after gravity: \n\n");
Initialize_matrix(matrix);
Update_rect(l_rect);
Update_matrix(matrix, l_rect);
Draw_matrix(matrix);

aux = l_rect;

while (aux)
{
    if (aux->y > 1)
    {
        good_to_go = EXIT_FAILURE;
        printf("\nError, the rectangle %d is not on the right position\n", aux->id);
    }
    aux = aux->next;
}

l_rect = Free_all_rect(l_rect);
return good_to_go;
}

int Stacking_rectangles(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
    LRect *l_rect, *aux;
    int i, good_to_go = EXIT_SUCCESS;

    printf("\n\n2nd Test, stacking rectangles:\n");
    Initialize_matrix(matrix);

    for (i = 1; i <= 8; i++)
    {
        s_coordinates->h = 3;
        s_coordinates->l = 5;
        s_coordinates->x = 1;
        s_coordinates->y = i * 3;
    }

```



```

if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
l_rect = Add_rect(l_rect, s_coordinates);
}

Update_rect(l_rect);
Update_matrix(matrix, l_rect);
Draw_matrix(matrix);

aux = l_rect;
i = 22;
while (aux)
{
if (aux->y != i)
{
good_to_go = EXIT_FAILURE;
printf("\nError, the rectangle %d is not stacked properly\n", aux->id);
}
aux = aux->next;
i -= 3;
}

l_rect = Free_all_rect(l_rect);

return good_to_go;
}

// Here we'll try a few
int Wrong_command(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
LRect *l_rect;
char *input;
int good_to_go = EXIT_SUCCESS;

l_rect = NULL;
printf("\n\n3rd Test, testing wrong commands:\n");

input = (char *)malloc(sizeof(char) * 20);
if (input == NULL)
{
printf("\nError while allocating memory (3rd test gravity, collison and commands)\n");
return EXIT_FAILURE;
}

printf("\n\n"create 1,1+3,4+1\n");
strcpy(input, "create 1,1+3,4+1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 1\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\n"create 1,1\n");
strcpy(input, "create 1,1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 2\n");

```

```

good_to_go = EXIT_FAILURE;
}

printf("\n\ncreate 1,1,1+3\n");
strcpy(input, "create 1,1,1+3");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 3\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\ncreate 1,1+3,3\n");
strcpy(input, "create 1,1+3,3");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 4\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\ncreate 1,1+1,2\n");
strcpy(input, "create 1,1+1,2");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 5\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\ncreate 1,1+2,1\n");
strcpy(input, "create 1,1+2,1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 6\n");
good_to_go = EXIT_FAILURE;
}

// Creating a rectangle for the next tests
s_coordinates->h = 5;
s_coordinates->l = 5;
s_coordinates->x = 1;
s_coordinates->y = 1;

if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
l_rect = Add_rect(l_rect, s_coordinates);

printf("\n\nmoveleft 1,1+3,4\n");
strcpy(input, "moveleft 1,1+3,4");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 7\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\nmoveright 1,1+3,4\n");
strcpy(input, "moveright 1,1+3,4");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{

```

```

printf("\nError while creating, input 8\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\nmoveright 1,1,3\n");
strcpy(input, "moveright 1,1,3");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 9\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\nmoveright 1,1,3+1\n");
strcpy(input, "moveright 1,1,3+1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 10\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\nmoveright 1+1+1\n");
strcpy(input, "moveright 1+1+1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 11\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\nmoveright 1,1\n");
strcpy(input, "moveright 1,1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 12\n");
good_to_go = EXIT_FAILURE;
}

printf("\n\nmoveright 1,1+1+1\n");
strcpy(input, "moveright 1,1+1+1");
if (Command_verify(input, s_coordinates) == EXIT_SUCCESS)
{
printf("\nError while creating, input 13\n");
good_to_go = EXIT_FAILURE;
}

free(input);
l_rect = Free_all_rect(l_rect);
return good_to_go;
}

int Collision_test(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
LRect *l_rect;
int i, collisions = 0;
int x[5] = {1, 1, 11, 11, 6};
int y[5] = {1, 4, 1, 4, 1};
int h[5] = {3, 3, 3, 3, 6};

```

```

l_rect = NULL;
printf("\n\n4th Test, collisions:\n");
Initialize_matrix(matrix);

/*creating 5 rectangles
which should give us 4 collisions 2 in the right, 2 in the left
*/
for (i = 0; i < 5; i++)
{
    s_coordinates->x = x[i];
    s_coordinates->y = y[i];
    s_coordinates->l = 5;
    s_coordinates->h = h[i];
    if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
    {
        l_rect = Add_rect(l_rect, s_coordinates);
        collisions += Collision_warning(l_rect, l_rect);
    }
}

Update_matrix(matrix, l_rect);
Draw_matrix(matrix);

l_rect = Free_all_rect(l_rect);

if (collisions != 4)
return EXIT_FAILURE;

return EXIT_SUCCESS;
}

int Efolio_B_Test(SCoordinates *s_coordinates, int matrix[MAX_LINES][MAX_COL])
{
    LRect *l_rect;
    int collisions = 0, i;
    int x[4] = {1, 5, 22, 21};
    int y[4] = {1, 6, 10, 12};
    int l[4] = {15, 12, 6, 6};
    int h[4] = {5, 3, 9, 7};

    l_rect = NULL;
    /*
    create 1,1+15,5
    create 5,6+12,3
    create 22,10+6,9
    create 21,12+6,7
    moveleft 23,11+2
    Moveright 10,8+2
    Delete 23,2

    1 collision at the end
    */

    for (i = 0; i < 4; i++)
    {

```

```

s_coordinates->x = x[i];
s_coordinates->y = y[i];
s_coordinates->l = l[i];
s_coordinates->h = h[i];
if (Valid_action(l_rect, s_coordinates->x, s_coordinates->y, s_coordinates->h, s_coordinates->l, -10) == EXIT_SUCCESS)
{
l_rect = Add_rect(l_rect, s_coordinates);
collisions += Collision_warning(l_rect, l_rect);
}
Update_rect(l_rect);
}

s_coordinates->x = 23;
s_coordinates->y = 11;
s_coordinates->p = 2;
s_coordinates->command = 'l';
Move_rectangle(l_rect, s_coordinates);
Update_rect(l_rect);

s_coordinates->x = 10;
s_coordinates->y = 8;
s_coordinates->command = 'r';
Move_rectangle(l_rect, s_coordinates);
Update_rect(l_rect);

s_coordinates->x = 23;
s_coordinates->y = 2;
s_coordinates->command = 'd';
l_rect = Delete_rectangle(l_rect, s_coordinates);
Update_rect(l_rect);

collisions += Collision_warning(l_rect, l_rect);

Initialize_matrix(matrix);
Update_matrix(matrix, l_rect);
Draw_matrix(matrix);

l_rect = Free_all_rect(l_rect);

if (collisions != 1)
return EXIT_FAILURE;

return EXIT_SUCCESS;
}

```