



RUSH

THE SQUARES



Preliminaries



Language: C

The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.



- ✓ Don't push your `main` function into your delivery directory, we will be adding our own. Your files will be compiled adding our `main.c` and our `my_putchar.c` files.
- ✓ You are only allowed to use the `my_putchar` function to complete the following tasks, but **don't push it** into your delivery directory, and don't copy it in *any* of your delivered files.
- ✓ If one of your files prevents you from compiling with `*.c`, the Autograder will not be able to correct your work and you will receive a 0.



Segfault, bus error, floating exception are all grounds for disqualification!



Allowed function: `write`.

The goal of this project is to display a square on the screen. In each assignment, the squares will look differently (see below).

You have to write the `rush` function, which will be called by **our** main function, which will look like this:

```
void rush(int x, int y);

int main()
{
    rush(5, 5);
    return (0);
}
```

In case of error, the function should return and display `Invalid size\n` on the **standard error output** (the newline feed should of course be interpreted).



All assignments must be done.

Team work

Here are a couple of important rules to follow:

- ✓ The **team leader** must register his/her group for the oral presentation on the intranet.
- ✓ You must complete your oral presentation on Sunday, at the right time, and **with all of your partners**.
- ✓ Every member of the group should be fully aware of the work you will have completed. Each member will be questioned, and your group's grade will be based on the worst explanations.
- ✓ You have to do everything within your power to contact your partners; look at their intranet profile, find them on Facebook or by any other mean. **Excuses regarding group problems will not be accepted.**
If, after you have tried **everything**, and your partner is still unreachable, send an email to the local staff **ASAP**.

Assigment 1

Delivery: rush-1-1/*

Here are the awaited displays when calling `rush(5, 3)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
o---o
|   |
o---o
```

When calling `rush(5, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
o---o
```

When calling `rush(1, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
o
```

When calling `rush(1, 5)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
o
|
|
|
|
o
```

When calling `rush(4, 4)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
o--o
|  |
|  |
o--o
```



A test binary is available on the intranet. **Usage:** `./rush1-1 x y`

Assigment 2

Delivery: rush-1-2/*

Here are the awaited displays when calling `rush(5, 3)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
/ *** \
*       *
\ *** /
```

When calling `rush(5, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
*****
```

When calling `rush(1, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
*
```

When calling `rush(1, 5)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
*
*
*
*
*
```

When calling `rush(4, 4)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
/**\
*   *
*   *
\**/
```



A test binary is available on the intranet. **Usage:** `./rush1-2 x y`

Assigment 3

Delivery: rush-1-3/*

Here are the awaited displays when calling `rush(5, 3)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
ABBBB
B    B
CBBBB
```

When calling `rush(5, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
BBBBB
```

When calling `rush(1, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
B
```

When calling `rush(1, 5)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
B
B
B
B
B
```


When calling `rush(4, 4)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
ABBA
B  B
B  B
CBBC
```



A test binary is available on the intranet. **Usage:** `./rush1-3 x y`

Assigment 4

Delivery: rush-1-4/*

Here are the awaited displays when calling `rush(5, 3)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
ABBBB
B    B
ABBBB
```

When calling `rush(5, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
BBBBB
```

When calling `rush(1, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
B
```

When calling `rush(1, 5)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
B
B
B
B
B
```

When calling `rush(4, 4)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
ABBC
B  B
B  B
ABBC
```



A test binary is available on the intranet. **Usage:** `./rush1-4 x y`

Assigment 5

Delivery: rush-1-5/*

Here are the awaited displays when calling `rush(5, 3)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
ABBBB
B    B
CBBBB
```

When calling `rush(5, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
BBBBB
```

When calling `rush(1, 1)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
B
```

When calling `rush(1, 5)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
B
B
B
B
B
```

When calling `rush(4, 4)`:

```
Terminal
~/B-CPE-100> cc *.c; ./a.out
ABBC
B  B
B  B
CBB A
```



A test binary is available on the intranet. **Usage:** `./rush1-5 x y`

More ?

Oh... You're done?

It can be a good exercise to try and make a generic `rush_generic` function that can create every type of squares on demand.

Then your `rush` functions will contain only one line that is calling your `rush_generic` function with some parameters to generate the correct square.

That'll remove duplicates of codes (which is generally a bad practice).

{EPITECH}

