

Lab 9: Building a Custom Graphics Library (Static & Dynamic)

C Programming: Problem statement (5 marks) In real-world software development, code is rarely written in a single file. Instead, it is separated into modular, reusable components called libraries.

In this lab, you will build your very own lightweight graphics library called `libcanvas`. Instead of rendering pixels to a screen, your library will render ASCII characters to a 2D text grid. You will implement the drawing functions, compile your code into both a Static Library (`.a`) and a Shared/Dynamic Library (`.so`), and write a client program to test it.

Part 1: The Library (`libcanvas`) You must create a header file (`canvas.h`) and an implementation file (`canvas.c`). Your library must maintain an internal 2D character array representing the screen (maximum size 100x100).

You must implement the following exactly as specified:

```
void initCanvas(int rows, int cols, char bg_char);
```

Initializes the canvas with the specified number of rows and columns. Every "pixel" on the canvas should be filled with `bg_char`.

```
void drawPoint(int x, int y, char brush);
```

Draws a single character (`brush`) at coordinates (`x`, `y`). If the coordinates are outside the canvas bounds, the function should safely do nothing.

```
void drawHLine(int x, int y1, int y2, char brush);
```

Draws a horizontal line of `brush` characters on row `x`, from column `y1` to `y2` (both inclusive). If the coordinates are outside the canvas bounds, draw the line up to the maximum canvas dimension and ignore the outer part of the line.

```
void drawRect(int x, int y, int w, int h, char brush);
```

Draws a hollow rectangle. The top-left corner is at (`x`, `y`), with width `w` (number of columns) and height `h` (number of rows). The outline is drawn using the `brush` character. Draw the rectangle up to canvas dimensions and safely ignore any parts that fall outside the canvas bounds.

```
void printCanvas();
```

Prints the entire canvas to STDOUT, row by row, with a newline at the end of each row.

Part 2: The Client Program (`main.c`) Write a client program `main.c` that includes your library using `#include "canvas.h"`. This program will read a sequence of commands from standard input (STDIN) to control your library. It should process commands until the End of File (EOF).

Input format: The first line will always be: `INIT <rows> <cols> <bg_char>` Subsequent lines will be the drawing commands in the following formats:

- `POINT <x> <y> <brush>`
- `HLINE <x> <y1> <y2> <brush>`
- `RECT <x> <y> <w> <h> <brush>`
- `PRINT` (This should call `printCanvas()`)

Sample Input/Output:

INPUT:

Plaintext

- `INIT 5 10 .`
- `POINT 2 2 *`
- `HLINE 0 1 8 =`
- `RECT 1 4 5 4 #`
- `PRINT`

OUTPUT:

Plaintext

- `.=====.`
- `....#####.`
- `..*#...#.`
- `....#...#.`
- `....#####.`

*(Explanation: The canvas is 5x10 filled with . . A * is at coordinates (2, 2). An = line runs across row 0 from column 1 to 8. # A hollow rectangle is drawn starting at row 1, column 4, with a width of 5 columns and a height of 4 rows.)*

Part 3: Compilation and Linking (The Core Task) You are required to write a single **Makefile** to automate the building of your project. Your Makefile must include the exact **gcc** and **ar** commands to handle both static and dynamic linking.

Your **Makefile** must contain the following specific targets:

- **static:**
 1. Compiles **canvas.c** into an object file (**.o**).
 2. Uses the **ar** utility to archive the object file into a static library named **libcanvas.a**.
 3. Compiles **main.c** and statically links it against **libcanvas.a** to produce an executable named **app_static**.
- **dynamic:**
 1. Compiles **canvas.c** into Position Independent Code (**-fPIC**).
 2. Creates a shared dynamic library named **libcanvas.so**.
 3. Compiles **main.c** and dynamically links it against **libcanvas.so** to produce an executable named **app_dynamic**.

Constraints

- $1 \leq \text{rows} \leq 100$ and $1 \leq \text{cols} \leq 100$.
- Coordinates are 0-indexed. The top-left corner is (0, 0).
- X-coordinates correspond to rows, Y-coordinates correspond to columns.
- You must NOT write a **main** function inside **canvas.c**.
- The state of the canvas must be hidden from **main.c** (use **static** global variables in **canvas.c**).

Grading Scheme (Total: 5 Marks)

- **1.0 Mark (Static Build):** Successfully generating the **libcanvas.a** static archive and linking it to produce a working **app_static** executable.
- **1.0 Mark (Dynamic Build):** Successfully generating the **libcanvas.so** shared library and linking it to produce a working **app_dynamic** executable.
- **3.0 Marks (Test Cases):** Your logic will be evaluated against 10 automated test cases (public and private), worth 0.3 marks each. (Note: The autograder will test both of your executables. Your final test case score will be the **maximum** score achieved by either your static or dynamic executable).

Extras (ungraded)

Extension 1: Arbitrary Line Drawing (Bresenham's Algorithm)

Currently, your library only supports horizontal lines (`drawHLine`). Real graphics libraries can draw a line between *any* two points. Task: Implement `void drawLine(int x1, int y1, int x2, int y2, char brush);` Hint: Look up and implement Bresenham's Line Algorithm. It is a classic algorithm that uses only integer addition, subtraction, and bit shifting (no expensive floating-point math like $y = mx + b$) to determine which pixels to fill to form a straight line between two arbitrary coordinates.

Extension 2: Circle Rasterization

Implement `void drawCircle(int cx, int cy, int r, char brush);` in your library. Use Bresenham's circle algorithm to mathematically calculate and draw a hollow ASCII circle on your grid.

Extension 3: Canvas Export (File I/O)

A graphics library isn't very useful if you can't save your masterpieces! **Task:** Add two new commands to your client program and library:

1. `void saveCanvas(const char* filename);` – Opens a file using `fopen()` and writes the 2D grid into it using `fprintf()`.
2. `void loadCanvas(const char* filename);` – Reads a previously saved text file and loads it back into your `grid` array, updating the rows and columns accordingly.

Command format for `main.c`:

- `SAVE my_drawing.txt`
- `LOAD my_drawing.txt`