Intro to HD61700 Programming

Chapter 06 Lab: Fun

1. Student Lab Assignment

Overview

Build a complete interactive **Brainfuck interpreter application** for the HD61700. The language is deliberately tiny; the work is in constructing the machine around it: editable source, interpreter state, loops, input/output, a finite data tape, a small program library, and a safe way back to the application menu.

For this lab, the specification below defines the course dialect. Do not depend on behavior not stated here. Brainfuck is used as a deliberately minimal interpreter target; its use for this course assignment has been cleared with the faculty.

Learning Path

The **core lab** demonstrates B-level mastery: editor, library, complete interpreter, bounded memory, input/output, loops, and graceful abort. The optional **A-level ROM** adds an execution debugger that can stop, expose interpreter state, step, and continue. The two implementations are delivered as **separate 64 KiB application ROM images**.

Application and Memory

Assume **32 KiB of contiguous RAM** is available to this application. The operating system and other machine state are outside this working budget for purposes of the lab.

Use the 32 KiB for:

- editable Brainfuck source;
- the Brainfuck data tape;
- editor/interpreter workspace and supporting structures.

Choose and document a sensible partition. Dynamic allocation is not required. The source area must hold the required library programs with useful editing headroom, and the tape must be large enough to run them. Memory regions must never corrupt one another.

The application should make its resource choices visible in a compact form, for example:
`SRC 146/4096  TAPE 16384`

The exact sizes and display layout are yours to choose.

Main Menu and Control

The core application provides at least:
`[ edit ] [ run ] [ library ]`

`MENU` is the application-level escape. From editing, library selection, or a running program, `MENU` returns to the Brainfuck main menu and preserves the current source buffer. During execution it abandons the current run state.

`BRK` retains its normal device meaning: reinitialize the machine. It is not a substitute for implementing `MENU` correctly.

Editor

The source is a **linear sequence of characters**; it has no logical concept of line. The editor presents a scrolling multi-line viewport onto that sequence. Display wrapping is presentation only and must not insert characters into the source.

The editor must provide sensible functional behavior for:

- cursor movement with the arrow keys;
- insertion at the cursor;
- deletion at the cursor;
- movement through source longer than the visible viewport;
- safe handling when the source buffer is full.

Insertion and deletion must preserve the rest of the source. If an insertion cannot fit, it must fail without corrupting the current program. The exact buffer-moving strategy, cursor representation, scrolling algorithm, and screen furniture are implementation decisions.

CS617 Brainfuck Specification

The interpreter recognizes exactly eight commands. Every other source character is a comment and has no effect on execution.

Command	Required behavior
`>`	move the data pointer one cell right
`<`	move the data pointer one cell left
`+`	increment the current cell
`-`	decrement the current cell
`.`	output the current cell as one character
`,`	read one character into the current cell
`[`	if current cell is zero, continue after matching `]`
`]`	if current cell is nonzero, continue after matching `[`

Additional rules:

- Tape cells are unsigned 8-bit values. `+` wraps `FF -> 00`; `-` wraps `00 -> FF`.
- The tape is finite. At the beginning of each new `run`, all cells are zero and the data pointer starts at the first cell.
- Moving left of the first tape cell or right of the last is a runtime error; terminate the run cleanly.
- Execution begins at the first source character and ends normally when the source position reaches the current source length.
- Loop nesting must work correctly. If execution requires a matching bracket and none exists, terminate cleanly with an error.
- `.` uses the course's normal character-display convention; printable ASCII values used by the official tests must appear as their corresponding characters.
- `,` waits for one keyboard character, stores its byte value in the current cell, and resumes execution.
- `MENU` must be checked often enough that even a non-terminating Brainfuck program can be abandoned without using `BRK`.

On normal completion, leave the final output visible until the user returns with `MENU`. Output longer than the visible area must remain usable by a sensible scrolling or windowing policy of your choice.

Library and Official Test Programs

`library` selects a ROM-resident example and **replaces the current edit buffer** with that source. After loading, it is ordinary editable source. At minimum, include the following CS617 conformance programs.

1. **Output A**
   ```
   ++++++++[>++++++++<-]>+.
   ```
   Expected output: `A`
2. **Nested loop**
   ```
   ++[>++++[>++++++++<-]<-]>>+.
   ```
   Expected output: `A`
3. **Echo one character**
   ```
   ,.
   ```
   Expected behavior: read one character and display the same character.
4. **Hello World**
   ```
   ++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.
   +++++++..+++.>++.<<+++++++++++++++.>.+++.------.
   --------.>+.
   ```
   Expected output: `Hello World!`

The line breaks shown in the handout are for readability only; they are not part of the required stored representation.

Optional A-Level Extension - Execution Debugger

Deliver a second ROM image containing the same application plus a debugger, for example:
`[ edit ] [ run ] [ library ] [ debug ]`

Debug mode starts from the same fresh execution state as `run`, but lets the user control execution. At minimum provide:

- execute one Brainfuck instruction;
- execute up to ten Brainfuck instructions;
- display enough state to identify the source/program position, current/next instruction, data-pointer position, and current cell value;
- preserve interpreter state between debugger steps;
- handle input, output, loops, normal termination, errors, and `MENU` consistently with normal execution.

Ignored/comment characters do not count as Brainfuck instructions for stepping. A ten-instruction step may stop early because the program terminates, reports an error, waits for input, or the user returns with `MENU`.

The debugger should expose and control the **same interpreter state**, not be a second unrelated interpreter implementation.

Apple Polisher Prize

The TA may award an **Apple Polisher** distinction for unusually clean modular assembly. Sensible includes, useful reuse, and clear boundaries among editor, interpreter, I/O, library, and debugger support are strong evidence; file count alone is not.

Functional deliverables: one 64 KiB core ROM image for B-level work; a second 64 KiB debugger ROM image for A-level work.

2. Teaching Assistant (TA) Guide & Cheat Sheet

Teaching Intent

The progression is **editable source** → **interpreter** → **observable execution**. Brainfuck keeps parsing trivial so that students must build the surrounding machine: source state, tape state, control flow, I/O, user escape, and a small but real editor.

Specify observable behavior and invariants. Do not turn the assignment into an HD61700 transcription. Buffer shifts, register allocation, loop-search method, viewport representation, tape placement, polling strategy, and module boundaries remain student work.

The B-level application is already substantial. Do not require bracket preprocessing, dynamic allocation, breakpoints, watches, undo, clipboard, search/replace, or a multi-line text model.

Memory and Editor Checkpoints

The 32 KiB budget is intentionally open. A fixed partition is fully acceptable. Allocating source/workspace from one end and tape from the other is also a reasonable design, but must not be prescribed as the only solution.

A good B submission can explain:

- source capacity and current source length;
- tape capacity;
- workspace/support use;
- why the regions cannot overlap;
- what happens at a full source buffer or tape boundary.

The editor has no logical lines. The important conceptual separation is:

```
source buffer    actual program in RAM
cursor position  offset into that source
viewport         visible window onto the source
```

Insertion should make following characters appear to move right; deletion should make them close left. The implementation mechanism is deliberately unspecified. Test edits near the beginning, middle, end, viewport edge, and full-buffer boundary.

Interpreter Reference

For each new `run`: preserve source, clear the tape, set the data pointer to its first cell, set source position to the first source character, and reset transient output/interpreter state as appropriate.

Only `><+-.,[]` have semantics. Other characters are skipped silently. Nested brackets must pair correctly when used. A straightforward B implementation may scan source at runtime to find a matching bracket; preprocessing or a jump table is welcome but not required and does not by itself earn the A-level designation.

Important behavioral tests:

- empty source terminates normally;
- comments/non-command characters are ignored;
- `FF + 1 -> 00` and `00 - 1 -> FF`;
- left and right tape limits produce clean errors;
- nested loops work;
- an encountered unmatched bracket produces a clean error;
- comma pauses for input and resumes;
- the four official library programs produce the stated results;
- repeated `run` begins from a fresh zeroed tape;
- `MENU` returns cleanly from edit, library, normal output, input wait, and active execution.

The critical non-termination test is:
`+[]`

It should run indefinitely until `MENU` is pressed, then return to the Brainfuck main menu without requiring `BRK`. A submission that can escape only by reinitializing the device has not met the application-level abort requirement.

Official Library Results

The required examples provide a shared conformance baseline rather than a general Brainfuck anthology.

Program	Main feature	Required result
Output A	basic loop/output	`A`
Nested loop	nested bracket semantics	`A`
Echo	input/output	input character echoed
Hello World	integrated execution	`Hello World!`

Students may include additional examples. The required programs must still be present and load into the editable source buffer rather than bypassing it.

A-Level Debugger

The debugger earns A-level consideration because the student must expose and suspend the interpreter's execution state, not because another menu item exists.

The relevant persistent state normally includes at least:

```
source/program position
data pointer
tape contents
source buffer
I/O state as needed
execution/debug state
```

A single-step executes one *Brainfuck instruction*; ignored characters are scanned past without consuming the step. A ten-step operation executes at most ten Brainfuck instructions and then yields control with the resulting state still intact.

A strong test program is the required nested-loop example. The TA should be able to step it and see source position, data-pointer movement, cell changes, and loop control evolve coherently. `MENU` must leave debug mode gracefully and preserve the source buffer.

Do not require breakpoints, watchpoints, trace logs, tape editing, or arbitrary step counts. Those are enrichment, not the definition of A-level mastery.

B / A / Apple Polisher Distinction

Distinction	Evidence
B-level	complete conforming interactive interpreter application
A-level	same application plus controlled, inspectable execution debugger
Apple Polisher	exceptional modularity, reuse, and assembly craftsmanship

The Apple Polisher is orthogonal to A-level functionality. Look for natural include/module boundaries, small understandable interfaces, useful reuse between run/debug paths, a clear top-level composition, and little gratuitous duplication. Do not reward mechanical fragmentation.

Suggested Socratic Hints

Editor: "Is a screen row part of the program, or only one way of looking at the program?"

Insertion: "What must become true of every source character after the cursor when one new byte appears there?"

Loops: "If this bracket is nested, how do you know which opposite bracket actually matches it?"

Abort: "What if the interpreted program never reaches an input or output instruction? How will MENU still regain control?"

Memory: "Your application has 32 KiB. Which regions are fixed by your design, and what prevents them from colliding?"

Debugger: "What state must survive after one instruction so that the next instruction can continue exactly where execution stopped?"

Apple Polisher: "Could you replace the editor or debugger without untangling the interpreter itself?"