

Homework 1

Due Sunday, February 13 by 10:00pm

Handout 3
CSCI 334: Spring 2022

Turn-In Instructions

For this assignment, create one separate source code file for each question (e.g., `q1.epf`).

Turn in your work using the Github repository assigned to you. The name of the Github repository will have the form `cs334lab1_<your user name>`. For example, my repository would be `cs334lab1_dbarowy`. You should have received an invite to commit in the repository in your email. If you did not receive this email, please contact me right away!

Honor code: This is a partner lab. You may work with another classmate if you wish, and you may co-develop solutions. Remember: although you can work on code together, you must each independently write up and submit your solution. No code copying is allowed.

This assignment is due on Sunday, February 13 by 10:00pm.

Unix Accounts

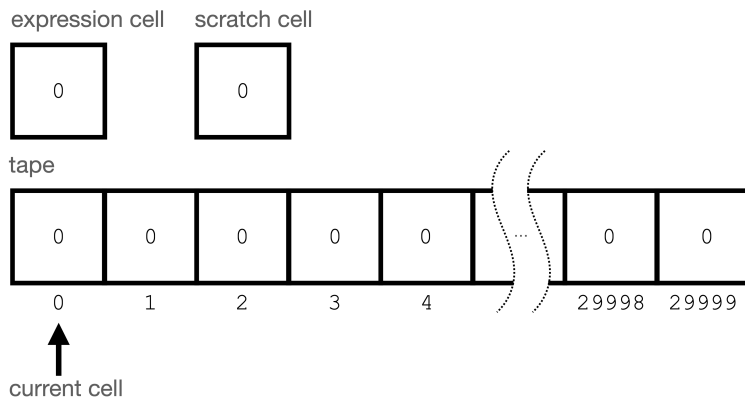
We will be working on the Unix lab computers throughout the semester. **If you have not used these machines before or don't remember your password, please see Mary Bailey in TCL 312 to obtain a password and verify that you can log in.**

I encourage you to work in the Unix lab whenever you like, but also keep in mind that you can `ssh` to our computers from anywhere on campus. For example, if your username is `bcool`, you can connect to `lohani` on the command line by typing: `ssh bcool@lohani.cs.williams.edu`

Breph Language Semantics

A Breph* machine consists of:

1. an array of 30,000 integer “cells,”
2. a pointer to the current cell, initially set to the first cell in the array,
3. an expression cell capable of storing one integer, initially containing 0, and
4. a “scratch” cell capable of storing one integer, initially containing 0.



*Pronounced “brief.”

When a user runs a Breph program, each operation in that program updates the Breph machine in a specific way. The following table describes Breph’s operations and their effects. Every Breph operation is an expression, meaning that evaluating it “returns” a value by storing it in Breph’s expression cell. $\langle e \rangle$ means “the value of the expression cell.”

Syntax	Meaning	Returns
i	Prompt user to enter a character c .	ASCII code for c
n	Prompt for a character of input and interpret it as a number n .	n
o	Print the result of $\langle e \rangle$ as a character.	$\langle e \rangle$
#	Print the result of $\langle e \rangle$ as a numeric character.	$\langle e \rangle$
+	Add one to the result of $\langle e \rangle$.	$\langle e \rangle + 1$
-	Subtract one from the result of $\langle e \rangle$.	$\langle e \rangle - 1$
*	“Dereference” operator.	the contents of the current cell
&	“Address of” operator.	the location of the current cell
.	Stores the result of $\langle e \rangle$ in the current cell.	$\langle e \rangle$
!	Changes the location of the current cell to the result of $\langle e \rangle$.	$\langle e \rangle$
s	Copies the value of the expression cell to the scratch cell.	$\langle e \rangle$
r	Copies the value of the scratch cell to the expression cell.	the value in scratch
\n	Returns 0.	0
j	If the current cell is zero, jumps to the operation after the next u.	$\langle e \rangle$
u	If the current cell is nonzero, jumps to the previous j.	$\langle e \rangle$
%	A line starting with % is a comment and is ignored.	n/a ; <i>removed from program</i>

Note that Breph reads input from *buffered I/O*. This means that when a user is prompted to enter input, they may enter as much or as little as they like. That input is stored in an array (a “buffer”), and as Breph needs to read input, it will read from the buffer. The following program demonstrates reading in three characters, storing them in consecutive memory cells, and then printing them back out.

```
i.
&+!i.
&+!i.
&--!*o
&+!*o
&+!*o
```

Here’s an example of running the above program, taking advantage of buffered input to enter a big string all at once. Notice that although we type in **danger**, Breph returns **dan**.

```
$ ./breph
danger      [we type this]
dan         [Breph prints this]
```

If you find yourself confused about how Breph interprets your program, run it in debugging mode, like so:

```
$ ./breph -d myprogram.eph
```

Debugging mode prints the machine’s state and pauses, waiting for you to press a key, before interpreting the current operation.

Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr
0	NULL	32	Space	64	@	96	`
1	Start of Header	33	!	65	A	97	a
2	Start of Text	34	"	66	B	98	b
3	End of Text	35	#	67	C	99	c
4	End of Transmission	36	\$	68	D	100	d
5	Enquiry	37	%	69	E	101	e
6	Acknowledgment	38	&	70	F	102	f
7	Bell	39	'	71	G	103	g
8	Backspace	40	(72	H	104	h
9	Horizontal Tab	41)	73	I	105	i
10	Newline	42	*	74	J	106	j
11	Vertical Tab	43	+	75	K	107	k
12	Form Feed	44	,	76	L	108	l
13	Carriage Return	45	-	77	M	109	m
14	Shift Out	46	.	78	N	110	n
15	Shift In	47	/	79	O	111	o
16	Data Link Escape	48	0	80	P	112	p
17	Device Control 1	49	1	81	Q	113	q
18	Device Control 2	50	2	82	R	114	r
19	Device Control 3	51	3	83	S	115	s
20	Device Control 4	52	4	84	T	116	t
21	Negative Acknowledgment	53	5	85	U	117	u
22	Synchronous Idle	54	6	86	V	118	v
23	End of Transmission Block	55	7	87	W	119	w
24	Cancel	56	8	88	X	120	x
25	End of Medium	57	9	89	Y	121	y
26	Substitute	58	:	90	Z	122	z
27	Escape	59	;	91	[123	{
28	File Separator	60	<	92	\	124	
29	Group Separator	61	=	93]	125	}
30	Record Separator	62	>	94	^	126	~
31	Unit Separator	63	?	95	_	127	Delete

Table 1: Numerical representations for the ASCII character set.

Problems

Q1. (10 points) Hello world!

Write a program in Breph that prints your Williams UNIX login. For example, my UNIX login is `dwb1`.

This program should use only the `*`, `+`, `-`, `o`, `#`, `.`, and `\n` operations (you may not even have to use all of these). Remember that the initial value of all memory cells of a Breph machine is 0.

Refer to Table 1, which will help you determine which stored numeric values correspond to what printed characters.

Call this program q1.eph.

Q2. (30 points) Shorter hello world!

Again write a program in Breph that prints your Williams UNIX login. However, this program should be half the length of the first program or less (not counting comments or whitespace). To achieve this, you may use any of the available operations except `i` and `n`.

Since generating the value of a given character is the expensive (in terms of the number of operations), a good strategy is to find a way to arrive at that number quickly. Multiplication, in the form of repeated addition, is one way to concisely generate a number.

For example, the following program will leave the number 35 in the second cell of its tape.

```
*****.  
j  
&+!  
*****.  
&-!  
*-.  
u
```

Call this program q2.eph.

Q3. (50 points) Calculator

Write a Breph program that prompts a user to enter two single-digit numbers, multiplies them, and then returns the result. For example, your program should operate exactly as follows when asking it to multiply 3 and 5:

```
$ ./breph q3.eph  
1: 3  
2: 5  
15
```

Note that because Breph uses buffered input, when a user types 3 and presses [Enter], there are actually two characters in the buffer, 3 and `\n`. If you plan to ask the user for more input, the trailing `\n` should be removed from the buffer first by asking Breph to read it (e.g., using the `i` command).

Call this program q3.eph.

Q4. (10 points) Language Design

As you may have gathered, writing these programs, Breph may not be appropriate for some programming tasks. Breph is inspired by a model of computation called a “Turing machine.” These questions ask you to think about the appropriateness of Turing machines for a given task.

- (a) One feature of a Turing machine is its simplicity. For the purpose Turing machines were invented, simplicity is an important feature. Simplicity should probably not be the most important feature for other problems. In particular, think about the problems I gave you on this assignment. If you could add one or more operations to Breph, what would you add? Choose one of your earlier solutions, and show how altering the language would simplify your program. Provide that simpler program.

You need not modify the Breph implementation yourself—this is a thought experiment. However, the proposed modification should not be obviously impossible.

- (b) Turing machines are better used for a specific purpose. What is that purpose? You will need to do a little research on your own to find an answer.

(Hint #1: you might start by trying to understand what the “Entscheidungsproblem” is.)

(Hint #2: Wikipedia has a surprisingly good definition of the Entscheidungsproblem.)

(Hint #3: If you are curious, have a look at Turing’s 1936 paper “On Computable Numbers, with an Application to the Entscheidungsproblem.”)

Provide your answers in a file called `q4.txt`.

Q5. (1 point) Optional Feedback

How hard was this assignment on a scale of 1 to 5? (where 1 is easy and 5 is difficult).

Do you have any additional comments or feedback that you would like me to know?

Please supply your answer as a `feedback.txt` file.