Homework 6

Due Sunday, October 23 by 10:00pm

Handout 14
CSCI 334: Fall 2022

Turn-In Instructions

For each programming question, create a project directory. For example, the source directory for question 1 should be in a folder called "`q1`".

Turn in your work using the Gitlab repository assigned to you. The name of the Github repository will have the form `https://evolene.cs.williams.edu/cs334-f22/<YOUR_USERNAME>/lab06.git`. For example, if your CS username is `22abc1`, the repository would be `https://evolene.cs.williams.edu/cs334-f22/22abc1/lab06.git`.

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. **Be sure to tell me who your partner is** by committing a `collaborators.txt` file to your repository (5 points). Be sure to include this file even if you have no partner.

This assignment is due on Sunday, October 23 by 10:00pm.

Sanity Check: Students sometimes submit incomplete assignments, accidentally forgetting to run `git add` for all of their files. Fortunately, there is an easy way to make sure that this does not happen to you. Before you are done, `git clone` your repository to a new folder and then try building/running everything. It only takes a couple minutes and can spare you from headaches later on.

Reading

1. **(Required)** "A Brief Overview of C"
2. **(Required)** "Growing a Language" by Guy Steele

Problems

Q1. (10 points) .... What does this program print?

Your starter code comes with a program, `q1.eph`, that prints something. What does it print?

The `$` symbol in the instructions below signifies doing something in the console; do not type the `$`.

(a) In your starter code is an interpreter for the Breph programming language, written in C. You will need to compile this interpreter. In order for the following command to work, your computer must have both `clang` and `make` installed. Lab computers in TCL 312 have these tools preinstalled.

```
$ cd breph
$ make
```

(b) If you see no errors, run the program (`q1.eph`) with the interpreter.

```
$ cd ../q1
$ ../breph/breph q1.eph
```

(c) What is the output? Save it by running the `breph` tool using the UNIX shell redirection. The `>` symbol, when written after a command, redirects the output of a program from the console to a file. For example,

```
$ ../breph/breph q1.eph > q1-output.txt
```

will save the program's output to a file called `q1-output.txt`. Be sure to `git add` and `git commit` that file.

I encourage you to play with this language. To see what features it has, run the interpreter without any arguments; this will print a help screen. When in the `breph` directory, run

```
$ ./breph
```

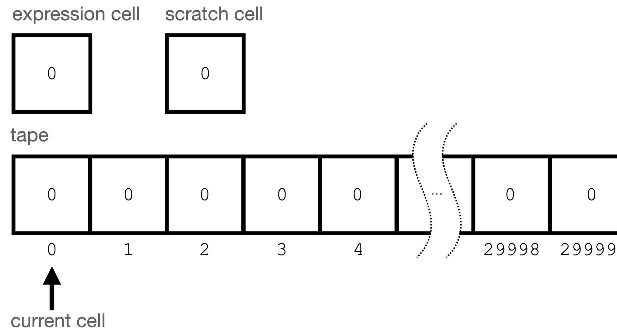
As you will see the syntax is simple—so simple that it's hard to see how the program above produces the output you see. Give it a try.

Your completed solution for this question should be called "`q1.eph`" and should be stored in a folder called "`q1`" in your repository.

Q2. (0 points) .... How Breph works

A Breph[*] machine consists of:

(a) an array of 30,000 integer "cells,"
(b) a pointer to the current cell, initially set to the first cell in the array,
(c) an expression cell capable of storing one integer, initially containing 0, and
(d) a "scratch" cell capable of storing one integer, initially containing 0.



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. $<e>$ 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 $<e>$ as a character.	$<e>$
`#`	Print the result of $<e>$ as a numeric character.	$<e>$
`+`	Add one to the result of $<e>$.	$<e>+1$
`-`	Subtract one from the result of $<e>$.	$<e>-1$
`*`	"Dereference" operator.	the contents of the current cell
`&`	"Address of" operator.	the location of the current cell
`.`	Stores the result of $<e>$ in the current cell.	$<e>$
`!`	Changes the location of the current cell to the result of $<e>$.	$<e>$
`s`	Copies the value of the expression cell to the scratch cell.	$<e>$
`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`.	$<e>$
`u`	If the current cell is nonzero, jumps to the previous `j`.	$<e>$
`%`	A line starting with `%` is a comment and is ignored.	*n/a; removed from program*

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
```

[*]Pronounced "brief."

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	123	z
27	Escape	59	;	91	[	124	{
28	File Separator	60	<	92	\	125	|
29	Group Separator	61	=	93	]	126	}
30	Record Separator	62	>	94	^	127	~
31	Unit Separator	63	?	95	_	128	Delete

Table 1: Numerical representations for the ASCII character set.

Q3. (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.

Your completed solution for this question should be called "`q3.eph`" and should be stored in a folder called "`q3`" in your repository.

Q4. (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
```

Hint: if you are totally stuck on how to make this program short, note that the previous question puts no limits on the length of the program.

Your completed solution for this question should be called "`q4.eph`" and should be stored in a folder called "`q4`" in your repository.

Q5. (40 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 q6.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).

Your completed solution for this question should be called "`q5.eph`" and should be stored in a folder called "`q5`" in your repository.

Q6. (10 points) .... **Helloworld**

The reading "A Brief Overview of C" demonstrates writing two things:

(a) a `helloworld` program, and

(b) a `Makefile`, used to compile the `helloworld` program.

Write these two programs and add them to your repository. Specifically, create a `helloworld.c` program and a `Makefile` that builds a `helloworld` target. Be sure to add both programs to your repository using `git add` and `git commit`.

For full credit, make sure that your `Makefile` can perform the following actions correctly:

- `$ make` should produce a compiled `helloworld` program, also producing `helloworld.s` file along the way.
- `$ make helloworld` should do the same thing as `$ make`.
- `$ make helloworld.s` should only produce the `helloworld.s` file.
- `$ make clean` should remove `helloworld.s` and `helloworld` files.

Your completed solution for this question should be stored in a folder called "`q6`" in your repository.