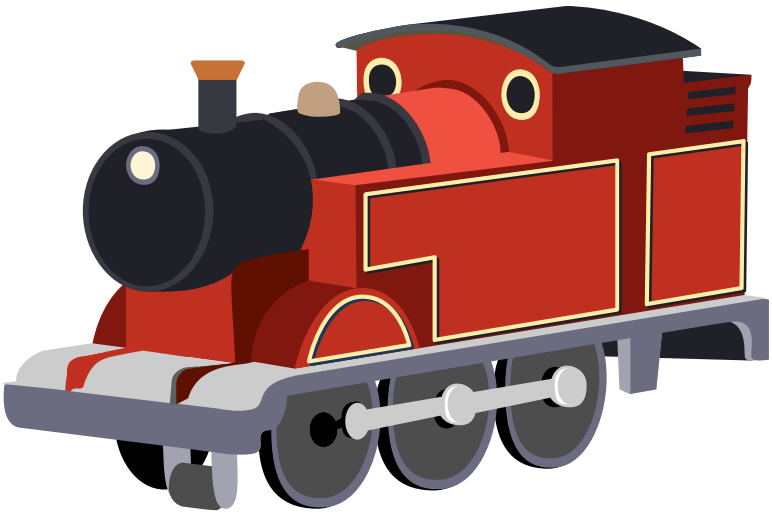
California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Fall 2022 – Lab 3 – *Hogwarts Express*

Overview



You have finally made it to King's Cross Station and, very discreetly, passed through the magical entrance at Platform 9 & 3/4. Before you, surrounded by wisps of steam and excited students, sits the scarlet Hogwarts Express.

But you don't have time to gape. The crowd of fellow students bustles you into the train. And, quite a short time later, you sit yourself – both excited and scared – into one of the many comfortable and ornate cabins.

Trees begin whip past your window as the train gathers speed. Where are you going? Where is Hogwarts? But, soon a more pressing question arises: Where is the food? You heard from some of the second-year students, that there is a witch that pushes a food trolley. Where is she?

Ka-chunk. Ka-chunk.

What was that? It seems like it is coming from the hallway.

Ka-chunk. Ka-chunk.

Something very odd walks by. It looks like one of the vending machines that you had in your elementary school. However, this one has two very large bucket-sized feet.

"Hello. The Trolly Witch has dragon-pox. I'm her replacement."

You are taken aback. A walking, let alone, a talking vending machine? This is one of the strangest things you have ever encountered. "Um, hi… okay… um… let me see what I want to eat."

It sighs. "Sorry, I can't help you."

"But, why? Are you… um…. broken." You are immediately afraid that you may have sounded a little bit rude.

It groans and shakes back-and-forth like it's nodding its head. "Kind of… my software isn't running correctly."

"Oh!" A feeling of excitement runs through your body. "I'm a muggle programmer! Maybe I can fix you!"

Your Task

You are going to use the odd muggle technology called "software" you fix the vending statue. Your program will display a menu of items, input the amount of knuts (wizard money), input their choice and then, output their selection and their change.

You must make use of tables. The user's input will act as an index. **Assume valid input.**

Example

Your solution doesn't have to look exactly like the example below. The user's input is printed in **blue**. The data outputted from your calculations is printed in **red**. You don't have to make the text that color in your program.

```
1. Bertie Bott's Every Flavor Sliders (23 knuts)
2. Cauldron Cakes (37 knuts)
3. Pumpkin Pasties (74 knuts)
4. Cancel the order (0 knuts)

Enter your selection:
2

You selected:
2. Cauldron Cakes (37 knuts)

How many knuts are you feeding it?
100

63 knuts is your change.
```

Part 1: Print Menu

Part 2: Get Selection

Part 3: Print Selection

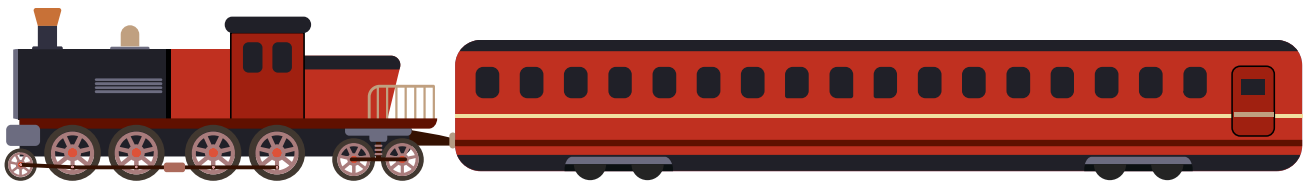
Part 4: Get Money

Part 5: Print Change

Have fun!

You can come up with your own theme. It doesn't have to be a food vending machine. Use your imagination!

- Cat items
- Dog items
- Rick and Morty items
- Pokemon items
- etc....



Tips for the Data Section

Here a few tips on how to structure your program.

- Create strings for each item in your vending machine.
- Create a table of those addresses (to lookup the purchased item).
- Also create a table of costs.

The following contains an example of how to make a table of addresses (which are quads) and table of values (also quads). Please feel free to change your labels.

```
Bertie:
   .ascii "1. Bertie Bott's Every Flavor Sliders (23 knuts)n\0"

Cake:
   .ascii "2. Cauldron Cakes (37 knuts)\n\0"
   …

Names:
    .quad Bertie
    .quad Cake
    …

Costs:
   .quad 23
   .quad 37
   …
```

Tips for the Text Section

- **DO ONE PART AT A TIME!** Once you have done it, assemble and debug. This is called "incremental design"
- The user is entering a 1-indexed number. This means the first item (for the student) is 1. However, this is the index 0 in the table. **Subtract 1.** Put the user's input into another register. You'll use this for indexing.
- Remember to use the correct scale for quads!
- Since the `Names` table already contains addresses, you should use MOV rather than LEA. Why? PrintZString needs an address and those are the values in that table. You want the address, not the address of the address.

Requirements

You must think of a solution on your own. The requirements are as follows:

1. Put your name in a comment at the top of your program.
2. Display a menu of items and costs. You must have (at least) three items. The last one should cost zero – which will let the user get back their money.
3. Input their selection
4. Output the item they bought to the screen. You must use a table.
5. Input how much money they entered
6. Output their change to the screen. You must use a table to look up the cost.
7. Put a **full** line comment that marks each part. Comments are your friend… and vital to good assembly programming.

Submitting Your Lab

To submit your lab, you must run Alpine by typing the following and, then, enter your username and password.

```
alpine
```

To submit your lab, send the assembly file (do not send the a.out or the object file to:

```
dcook@csus.edu
```

UNIX Commands

Editing

Action	Command	Notes
Edit File	**`nano`** *filename*	"Nano" is an easy to use text editor.
E-Mail	**`alpine`**	"Alpine" is text-based e-mail application. You will e-mail your assignments it.
Assemble File	**`as -o`** *object source*	Don't mix up the *objectfile* and *asmfile* fields. It will destroy your program!
Link File	**`ld -o`** *exe object(s)*	Link and create an executable file from one (or more) object files

Folder Navigation

Action	Command	Description
Change current folder	**`cd`** *foldername*	"Changes Directory"
Go to parent folder	**`cd ..`**	Think of it as the "back button".
Show current folder	**`pwd`**	Gives the current a file path
List files	**`ls`**	Lists the files in current directory.

File Organization

Action	Command	Description
Create folder	**`mkdir`** *foldername*	Folders are called directories in UNIX.
Copy file	**`cp`** *oldfile newfile*	Make a copy of an existing file
Move file	**`mv`** *filename foldername*	Moves a file to a destination folder
Rename file	**`mv`** *oldname newname*	Note: same command as "move".
Delete file	**`rm`** *filename*	Remove (delete) a file. There is **no** undo.