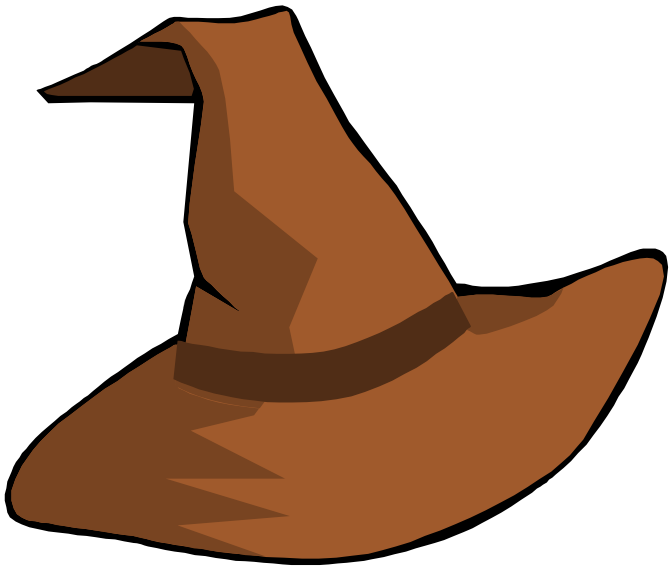
California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Spring 2021 – Lab 4 – *Sorting Hat*

Overview

Student wizards and witches are taught at the famous ***Hogwarts School of Witchcraft and Wizardry***. When students first arrive at this World-renowned school, they are sorted into one of the four different houses. This is accomplished with an ancient hat called the Sorting Hat.



You might belong in Gryffindor,
Where dwell the brave at heart,
Their daring, nerve, and chivalry
Set Gryffindors apart;

Or yet in wise old Ravenclaw,
if you've a ready mind,
Where those of wit and learning,
Will always find their kind;

You might belong in Hufflepuff,
Where they are just and loyal,
Those patient Hufflepuffs are true
And unafraid of toil;

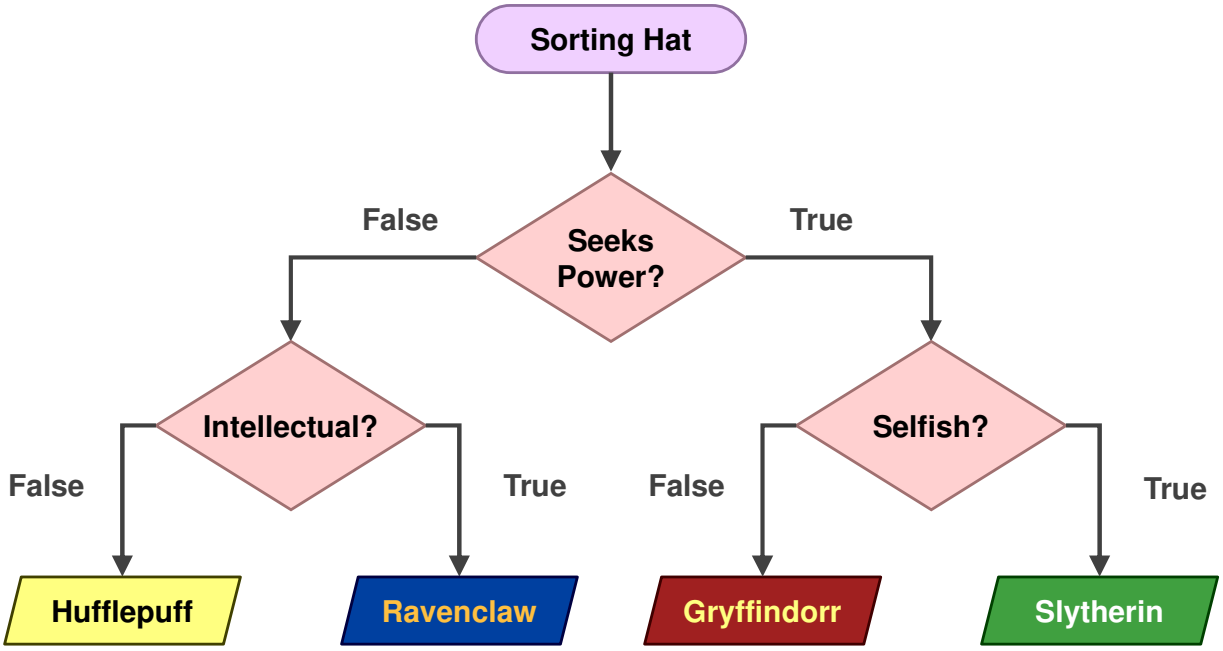
Or perhaps in Slytherin
You'll make your real friends,
Those cunning folks use any means
To achieve their ends.

So, how does this magical relic work? What logic does it use to sort each student?

The Sorting Hat's Algorithm

While the Sorting Hat seems mysterious and powerful, it follows a basic nested If Statement. Well, this isn't really true, but it's your instructor's attempt to turn it into a programming assignment.

So, for this assignment, let's **assume** that it simply senses the answers to a few questions and then puts the student into the correct house (hopefully). The logic is as follows:



Have Fun!

You don't have to use the four houses from Harry Potter. Create your own four categories. Naturally, the questions will be different, but the overall approach is the same. The following are some example categories.

- What class in a role player game are you?
- What should you have for dinner?
- What meme are you most resemble?
- Which SpongeBob character are you?
- Which Rick and Morty alien are you?
- What type pet should you get – species, breed, etc...
- What political philosophy are you?
- What music you should listen to?
- etc...

Your Task

So, for your assignment, you get to create a program to recreate the Sorting Hat. You will ask questions and the user will respond with a numeric value. Based on their answers, you will print which house they will be put into.

Examples

This is the output from one possible solution. Your solution doesn't have to look exactly like the example below. But, make sure to fulfill all the requirements. Input is displayed in **blue**.

```
Welcome to Hogwarts!
The Sorting Hat is being placed upon your head.

Do you wish to command/control others (y/n)?
n
Do you find contentment in reading (y/n)?
y
Ravenclaw!
```

```
Welcome to Hogwarts!
The Sorting Hat is being placed upon your head.

Do you wish to command/control others (y/n)?
y
Is it better to help yourself more than others (y/n)?
y
Slytherin!
```

```
Welcome to Hogwarts!
The Sorting Hat is being placed upon your head.

Do you wish to command/control others (y/n)?
y
Is it better to help yourself more than others (y/n)?
n
Gryffindor!
```

Tips

Reading Characters

CSC 35 Library has a subroutine called "ScanChar" that will read a byte from the keyboard and store it into `bl`.

```
call ScanChar              #bl = input;
```

How to approach the problem

- When a character is stored in `bl`, it will contain an ASCII character code. You will need to use these in your `cmp` instructions.
- Like all labs, **build it in pieces**. First get a single If-Statement to work. Then, you can work on to more detailed ones.
- All labels **must** be unique. Choose your names well.
- Assembly doesn't have blocks… so don't think in those terms. Your program will be structured far differently from programs in Java.

Requirements

Now work on each of the requirements below one at a time. You will turn in the final program, but incremental design is best for labs. You must think of a solution on your own.

Do **not** use direct storage (which we haven't covered yet). Any lab using direct storage will receive a zero.

The requirements are as follows:

1. Put your name and section # in a comment at the very top of your program. We are using class accounts, and I need to be able to identify you. *(5 points)*
2. Display text explaining that theme of your program. You don't have create the Hogwarts one that I used. Use your imagination. *(5 points)*

   telling the user what they are being sorted into (Hogwarts, party, etc…)
3. Input each choice (with prompts). *(10 points)*
   You **must** input a character. Any lab that uses ScanInt will receive a zero.
4. Implement the logic in the flowchart above. Use nested ifs *(10 points)*
5. Display the output for all four possibilities. *(10 points)*

Submitting Your Lab

This activity may only be submitted in Intel Format.
Using AT&T format will result in a zero. Any work from a prior semester will receive a zero.

Afterwards, run Alpine by typing the following and, then, enter your username and password.

```
alpine
```

Please send an e-mail to yourself (on your Outlook, Google account) to check if Alpine is working. To submit your lab, send the assembly file (not **`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 *object* and *source* 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.