

## Laboratory 12 Hangman

Computer Science 240

The final lab is designed to help you with the programming project, the game of Hangman. You will probably not complete all the exercises in lab, but do as much as you can – completing all the tasks implements the game!

*Exercise 1:* Start with the *hmskel.asm* file distributed with the project specification. The main program should welcome the user, and prompt them to enter a large value (such as the time of day). This value ‘seeds’ a pseudo-random number generator. The value entered should be stored in the variable **seed**. The main program should then call a procedure called **getindex**.

The procedure **getindex** should return a value to the main program which is an index into the table of 117 possible game words. The *hmskel.asm* file contains a function **rand** which uses **seed** to return a number between 0 and 32648 in \$v0 – **rand** should be called by **getindex** to produce the index.

The main program should then print the index, and prompt the user to ask them to play again. The console should look something like this:

```
Welcome to Hangman! Please enter the time (for
example, at 12 seconds past 8:30, enter 83012): 83012
```

```
112
```

```
Would you like to play again? ('y' or 'n'): y
```

```
98
```

```
Would you like to play again? ('y' or 'n'): y
```

```
14
```

```
Would you like to play again? ('y' or 'n'): n
```

```
Goodbye! Thanks for playing Hangman
```

Demonstrate to the instructor.

*Exercise 2:* Copy the procedure **getname** that you wrote for the lab assignment to your program. You may recall that you prompted the user for an index to be used by **getname** to select a name from the table. Now, use the index produced by the **getindex** procedure to have **getname** select a name from the game table of words. Print the selected word and its length from the main program, as you did last week.

Demonstrate to the instructor.

*Exercise 3:* Declare a variable **guessname** in the data section which will contain a string of characters (you must allocate enough bytes to hold all the characters in the longest possible name).

In your main program, write code to initialize **guessname** to all dashes, and output **guessname** after you output the name and its length:

**jean 4**

----

Demonstrate to the instructor.

*Exercise 4:* Add a prompt to your program which asks the user to input a single letter.

Search the name for the letter (you will need a way to handle letters being entered in either upper or lowercase).

If found, output the string **guessname**, but with the position representing the letter filled in with that letter. For example, assuming the name is **jean**:

**Enter letter: a**

--a-

If the letter is not found, output an error message.

Loop back to prompt the user to guess another letter.

Demonstrate to the instructor.

*Exercise 4:* When the player makes an incorrect guess, your program must output a string representing a portion of the hangman figure. The number of parts depends on how many errors (incorrect guesses) have been made by the player. The whole hangman figure is output when 7 incorrect guesses have been made.

The complete hangman figure can be represented by a character string **gallows**, whose data declaration is given below:

```
gallows:      .asciiz      "\n__\n |\n 0\n \\/\n / \\\n"
```

Be sure you understand how these ascii bytes form the hangman figure before you continue.

Write a new program (separate from what you have implemented so far) which prompts the user for a value of 1-7. Use this value to output the appropriate part of the hangman figure. You shouldn't have to define any other gallows strings than the one given above.

Demonstrate to the instructor.

*Exercise 5:* Add your code from exercise 4 to your main program. Instead of prompting the user for a value 1 - 7, keep a count of the number of incorrect guesses made. Use this value to output the hangman figure.

Demonstrate to the instructor.