Shakespeare Programming Language

Introduction

Bored late on a February evening while avoiding their homework assignment, Karl Hasselström and Jon Åslund developed The Shakespeare Programming Language. Their goal -- to write a language with source code that looked like a Shakespearean play. This language contains nothing fancy, just your everyday arithmetic and goto statements. In their own words, they have "combined the expressiveness of BASIC with the user-friendliness of assembly language."[1]

For the purposes of this interactive workshop, you will want to be able to access the following online compiler: Try It Online.

A Code by Any Other Name

During this workshop we will be taking a look at a SPL program that will read in input from the user character by character, have one of the actors store that information using a **stack**[2], and then have that actor output the information they have been storing in reverse order. For example, if our input is *!dlroW olleH,* the output will be *Hello World!* To put this in more Shakespearean terms, we will be *Outputting Input Reversedly*.

With this in mind, let the next sections act as a primer on writing a program in the Shakespeare Programming Language (SPL).[3]

Title

Everything up until the first period of an SPL program is the title. The title is merely aesthetic, serving no real purpose in the code except to **comment**[4] what the program will be about.

Dramatis Personae

You must list all of the characters, or **variables**[5], you want to use in your SPL program at the beginning, just under the title. You may use virtually any of the myriad characters available within the works of

[1] The Shakespeare Programming Language, Karl Hasselström and Jon Åslund, 21 August 2001.
[2] In computing, a stack is a data structure used to store a collection of objects. Individual items can be added and stored in a stack using a PUSH operation. Objects can be retrieved using a POP operation, thereby removing an item from the stack.
[3] Should you be so inclined, you can check out the full documentation here.
[4] A comment is a line of code ignored by the parser/compiler used for informational purposes by programmers.
[5] Variables are simply containers used to store values that can be altered during program execution.

Shakespeare. SPL only deals in signed integer values, in other words, negative and positive whole numbers.

Acts and Scenes

These divide our *play* into smaller parts. They act similar to **functions**[6] in other programming languages. Acts and Scenes are numbered using ROMAN NUMERALS, begin with the word *Act* or *Scene*, followed by the number, and then a description of the act or scene.

Enter, Exit, and Exeunt

In order to interact with each other, characters must be *on-stage.* Enter, Exit, and Exeunt cause characters to get on and off the stage. *Enter* is followed by a list of one or more characters. *Exit* is followed by exactly one character and *exeunt*, the plural form of exit, is followed by a list of at least two characters, or no characters, in which case all characters exit the stage. You can think of *Exeunt* as program termination, generally called only at the end of the play.

Lines

Lines consist of a character name followed by a colon and typically contain one or more sentences. In the program for this workshop, we will use six kinds of sentences: *input*, which causes the character to require information from the user; *output*, which causes output to the screen; *gotos*, which cause the characters to return to a previous scene or act; *conditional statements*, which cause the characters to make a choice; *pushing/popping*, which causes a character to store/remove information; and *statements*, which cause the specified character to assume a certain value.

Constants[7]

In SPL, **nouns**[8] have a constant value of 1 or -1, depending on whether it is *nice* or not. *Nice* and *neutral* nouns will have a constant value of 1, while ***!nice***[9] nouns have a value of -1. By prefixing a noun with an **adjective**[10], you multiply it by two. Increasing the number of adjectives before a noun will thereby increase the value of the noun by a power of two. You can use the following formula to calculate a value:

$$value = noun \times 2^{\#-of-adjectives}$$

[6] Functions are self-contained modules of code that accomplish a specific task. It takes in something, processes that input, and then returns a result.
[7] A constant in programming is a container with a value that cannot change during execution.
[8] A noun is a word used to identify any of a class of people, places, or things, or to name a particular one of these.
[9] In many programming languages the "!" stands for "not"; thus, !nice can be read as "not nice".
[10] An adjective is a word or phrase naming an attribute, added to or grammatically related to a noun to modify or describe it.

Let us take for example, "the difference between the square of the difference between my little pony and your big hairy hound and the cube of your sorry little goat." This would be equivalent to the following equation:

$$(my\ little\ pony\ -\ big\ hairy\ hound)^2\ -\ (sorry\ little\ goat)^3$$

Which in turn, when we plug in the constant values using our above formula, we get:

$$((1\ \times 2^1)\ -\ (1\ \times 2^2))^2 - ((-1)\ \times 2^2)^3\ = (2\ -\ 4)^2 - (-4)^3 = 4 - (-64)\ = 68$$

As you can see, this method of writing constants gives you much more poetic license than most other programming languages.

Assignment of Values

Okay, so now that we have some numbers, just exactly how do we use them? Let's take a look at the following sentence: "You lying stupid fatherless big smelly half-witted coward!" Here we have a second-person pronoun, followed by a number. The effect of this statement is to assign the value **(-64)**[11] to the character being spoken to.

There are few other ways to input, but we will discuss those at greater length in the sections which pertain directly to them.

Output

There are two different types of output sentences, *Open your heart* and *Speak your mind.* The first one, *Open your heart*, will cause the character being spoken to to output their value in numerical form. The second one, *Speak your mind*, will cause the character being spoken to to output their value as the corresponding letter, digit, or character in **ASCII**[12].

Input

The input statements work very much like the output statements, except they read information into the program instead of writing information as output. To read a character, use the phrase *Open your mind.* To

[11] "You lying stupid fatherless big smelly half-witted coward" has 6 adjectives and the noun "coward" is !nice, and therefore has a constant value of -1. Thus this statement comes to -1 x 2^6 = -64.

[12] ASCII stands for the American Standard Code for Information Interchange, and it is a character encoding standard for electronic communication. A numerical value (0-127) is assigned to each letter, digit, character, and non-printing character as computers only understand numbers … not English): If you find this interesting, and you just might as you are down here reading the footnotes, check out some fun things you can do with character encoding.

read in an integer, use the phrase *Listen to your heart.* As you might have guessed, the value will be assigned to the character being spoken to.

Gotos[13]

A sentence like "Let us return to scene III" simply means *goto scene III.*

Conditional Statements[14]

Conditional statements in SPL come in two easy steps. Let's take a look at the following code fragment to see how they work.

```
Juliet:
  Am I better than you?

Hamlet:
  If so, let us proceed to scene III.
```

Here we see Juliet pose a question. Behind the scenes, the **parser[15]** checks the value currently being held by Juliet to see if it is *greater than* the value currently being held by Hamlet. If the value currently being stored in Juliet is indeed greater than the value currently being stored by Hamlet, the **program counter[16]** will advance to scene III.

Comparisons

Comparisons, or the examination to determine if two things are similar, different, or equal, are constructed as follows. The statement "is X as good as Y" tests for equality. To check to see if one value is *greater than* another value, use a *nice* (or positive) comparative word, "is X better than Y." To check to see if one value is *less than* another value, use a *!nice* comparative word, "is X worse than Y."

[13] A goto statement performs a one-way transfer of control to another line of code. In other words, it's bossy (one might even say shrew-ish) and tells the code which line to run next.
[14] Conditional statements are used to make decisions based on some condition. For example, *if* it is raining outside, I will grab my umbrella, *otherwise* I will not grab my umbrella.
[15] A parser is a software component that takes input data (frequently text) and builds a data structure -- usually a kind of tree -- and then uses that data structure to define what the input means in the context of the program.
[16] A program counter is a register in a computer processor that contains the address (or location) of the next instruction to be executed.

Stacks

Characters in SPL are not simple-minded characters. They are not limited to the ability of only remembering a single number. Much like you or I, they can remember several numbers (except for phone numbers … everyone is bad at remembering those). Character memories are implemented with stacks.

Every character can *push* (add a value to their memory) and *pop* (remove a value from their memory). Pushing is performed like this:

```
Lady Macbeth:
  Remember me.
```

The above line of code will force whoever Lady Macbeth is speaking to to push the value stored by Lady Macbeth onto their stack (or into their memory, if you will).

Popping (removing a value from their memory) is performed like this:

```
Lady Macbeth:
  Recall your imminent death!
```

Only the word "recall" is important here; everything else is just poetic license. This piece of code causes whoever Lady Macbeth is talking to to pop an integer from their stack and assume that value for themself.

Our Example Program

On the next page, I will outline our input reversing program. SPL does not have inline comments, so please note that any *blue italicised text* should be thought of as a comment, and not as compilable SPL code. I will use this special blue italicised text to let you know what is going on in the program at the time, so that you can use this sample program as a launching board for further experiments in this language … or any programming language you choose to pursue.

Outputting Input Reversedly.

Juliet, a stacky mortal.
Puck, a mischievous fairy who delights in pushing mortals until they pop.

Act I: The one and only.
Scene I: In the beginning, there was nothing.

[Enter Puck and Juliet]
Juliet:
  You are nothing! *Sets the value of Puck to zero.*

Scene II: The Pushing of Poor Juliet

Puck:
  Open your mind! Remember yourself. *Reads ASCII input, stores on stack*

Juliet:
  You are as tiny as the sum of yourself and a hamster. *Adds one to Puck*
  Am I as miserable as a blister? *Checks for end of input*

Puck:
  If not, let us return to scene II. *Return scene II if not at end of input*
  Recall your playful nature! *Removes end of input character from Juliet*

Juliet:
  You are as distasteful as the difference between yourself and a summer's day.
  Reduces the value of Puck by one

Scene III: Once you pop, you can't stop!

Puck:
  Recall your unhappy arrangement. *Pop a value from Juliet's memory*
  Speak your mind! *Print the ASCII character to the monitor*

Juliet:
  You are as foul as the sum of yourself and a plague! *Reduces Puck by one*
  Are you fresher than nothing? *Checks to see if Puck is holding a zero*

Puck:
  If so, let us return to scene III. *If Puck not zero, return to scene III*

Scene IV: The end.
[Exeunt] *All characters leave the stage and the program ends*

List of Nouns

Positive Nouns (+1)	Neutral Nouns (+1)	Negative Nouns (-1)
Heaven	animal	Hell
King	aunt	Microsoft
Lord	brother	bastard[17]
angel	cat	beggar
flower	chihuahua	blister
happiness	cousin	codpiece
joy	cow	coward
plum	daughter	curse
summer's day	door	death
hero	face	devil
rose	father	draught
kingdom	fellow	famine
pony	granddaughter	flirt-gill[18]
	grandfather	goat
	grandmother	hate
	grandson	hog
	hair	hound
	hamster	leech
	horse	lie
	lamp	pig
	lantern	plague
	mistletoe	starvation
	moon	toad
	morning	war
	mother	wolf
	nephew	
	niece	
	nose	
	purse	
	road	
	roman	
	sister	
	sky	
	son	
	squirrel	
	stone wall	
	thing	
	town	
	tree	
	uncle	
	wind	

[17] See footnote 18.
[18] By far the worst of the !nice nouns. Remember, words define us, they explain us, and on occasion, they serve to control or isolate us. Always use your words wisely.

List of Adjectives

Positive Adjectives	Neutral Adjectives	Negative Adjectives
amazing	big	bad
beautiful	black	cowardly
blossoming	blue	cursed
bold	bluest	damned
brave	bottomless	dirty
charming	furry	disgusting
clearest	green	distasteful
cunning	hard	dusty
cute	huge	evil
delicious	large	fat
embroidered	little	fat-kidneyed[19]
fair	normal	fatherless
fine	old	foul
gentle	purple	hairy
golden	red	half-witted
good	rural	horrible
handsome	small	horrid
happy	tiny	infected
healthy	white	lying
honest	yellow	miserable
lovely		misused
loving		oozing
mighty		rotten
noble		smelly
peaceful		snotty
pretty		sorry
prompt		stinking
proud		stuffed
reddest		stupid
rich		vile
smooth		villainous
sunny		worried
sweet		
sweetest		
trustworthy		
warm		

[19] gross, clumsy