Fall 2015 COMS 4115
Programming Languages & Translators
Project Proposal

Prof. Stephen Edwards   Due September 30, 2015
Columbia University   at 4:10 PM

[1]

StoryBook

Authors
Nina Baculinao (nb2406) - Systems Architect
Beth Green (blg2132) - Language Guru
Anna Lawson (aal2150) - Testing
Pratishta Yerakala (py2211) - Manager

[1] Image source: http://wagomaha.com.s121312.gridserver.com/wp-content/uploads/2012/04/cat-reading-a-book-with-glasses-600x384.jpg

Introduction

StoryBook is a programming language targeted toward novice programmers who are just beginning to understand the basics of computer science and logical thinking. The language uses intuitive, "story-like" syntax and structure to make object-oriented programming easier for children and adult-beginners to read and implement.

The language uses the idea of a story where there are *characters* (objects), who have *traits* (instance variables) and can execute *actions* (methods). These characters can act in *chapters* (functions), and multiple chapters can come together in sequence to form the *plot* (main function) of a story. Just as one character in a story can pass critical information along to another, and what happens in one *chapter* might influence later events in a story, the subcomponents of a "story" program can communicate by sharing (returning) information about their "conclusions." This allows users to learn the basics of computer science, and more specifically, object-oriented programming, in the familiar, intuitive context of stories.

Colloquial words are used for reserved keywords instead of symbols that might not be intuitive to most (such as "and" instead of "&&," and "is" for assignment). Moreover, common symbols (such as "=") are used as they are in basic math and vernacular, rather than adhering to computer science conventions that may be counterintuitive to novices. This minimizes the syntax learning curve for beginners, allowing them to instead focus on mastering basic concepts of computer science and learning to think in a more logical way before moving on to more complex languages. StoryBook, by allowing users to implement basic algorithms using simple syntax and the familiar structure of stories, serves as an introductory programming language that can be a gateway to more complex languages and computer science concepts.

Motivation

Computer Science, programming especially, is quickly becoming a large part of the educational curriculum from grades K-12. However, many students are expected to simply jump into logical thinking via courses like AP Computer Science or haphazardly pick it up by reading endless threads on StackOverflow. We propose StoryBook, a programming language that focuses on readability and intuitiveness to help younger kids learn to think in a logical way and embrace computational thinking. It helps bridge the gap between passively making logical sentences and the deliberate steps it takes to use logic to solve a problem.

Syntax

Data Types

- We tried to stick to colloquial terms to make our language more intuitive for children and people less familiar with computer science.
- `number` - Everything is a decimal. No integer division. Number is more intuitive than integer, float, long, double, etc for children and people less experienced with programming.
- `letter` - Represents a single char. We used the more conversational term "letter" in order to distinguish a char from a character in a story.
- `words` - String
- `tof` - True or false (Boolean)
- `list[]` - Dynamically-sized. All values must be of the same type.
- `Character` - User defined object. Inheritance is implemented.
  - keyword `trait` - Instance variable.
  - keyword `action` - Class method.
  - keyword `subtype` - Inheritance.

Complete Statements

- To indicate a complete statement, we use `.` instead of the `;` commonly used in Java and C to make the statement syntax more "story-like" and more closely adhere to the conventions of English grammar.

Comments

- `~~one line comment`

-
  ```
  ~~block comment
  this is another line in the comment~~
  ```

Indentation

- Indentation should be regular. Striving for readability may lead to more verbose statements, so by convention use 2 spaces for indentation to save space, instead of 4.

Variable Declaration

- `number x is 5. ~~any variable`
- `trait number y is 6. ~~instance variable`
- We added the `trait` keyword for instance variables to help clarify that the instance variables pertain to a specific character (i.e. instance of an object).
- `letter list myList2 is ['a';'b';'c']. ~~list`
- `number list myList is []. ~~empty list`

Character Declaration

- Character object variable names are capitalized by convention
- `’s` accesses instance variables (example: `SleepingBeauty’s name`)
- `,` invokes a method on an object (example: `SleepingBeauty, wakeUp.`)
- Defining a Character type:
```
Character Soldier {
  words trait name.
  number trait age.
  words trait favoriteColor.

  Action Fight(number x) {
    ~~Do something.
  }
}
```
- Instantiating a `Soldier` Character object:
```
Soldier Jack is new Soldier(name is Jack; age is 15;
favoriteColor is "green").
```

Chapter Declaration

- Regular Functions
  - Regular functions are called Chapters, to follow the story-like structure
  - Declaring a function with parameters and a return value:
    `Chapter Sum(number x; word y) return number{ }`
    - `;` Semicolon separates function arguments
    - keyword `return` <type> denotes the return type
  - Calling a function: `number result = Sum(1; 2).`
  - keyword `return` is used for return statements at the end of a Chapter body
- Main Method
  - `Plot(word list args){ }` - Main function with command line args
  - `Plot(){ }` - Main function with no arguments

Print Statements

- `say` (example: `say("Hello World")`)
- `say` is a built-in function for every `Character` (example: `Wolf, say "I'll huff, and I'll puff, and I'll blow your house in!"`)
- For Characters only: `say(whoamI)` will generate a description of their traits and actions
  - `Jack, say(whoamI)` will print out
  - `Jack: I am a Soldier. My name is Jack, my age is 15, my favoriteColor is green. I can Fight.`

Basic Operators

- Variable assignment: `is` (example: `x is 5`).
- Testing equality: `=, >, <, >=, <=, and, or, isnot`
  - We used "=" to test for equality instead of "=="
- Mathematical operations: `+, -, /, *`
  - May add more math functions if the need arises
- Do not allow += or `++/--` because it is not intuitive. Must use `x is x + 1`
- String concatenation: `+`

List Operators

- `list1 + list2 ~~evaluates to new list of joined lists`
- `list1's length ~~evaluates to the length of the list`
- `list1[3] is 23. ~~changes the item at index 3 to 23.`
- `list1, append(data is 7). ~~appends number 7 to end`
- `list1, insert(data is 5; position is 1) ~~inserts at position 1 (the first item), shifting other list items`

Control Flow

- If else statement example:
```
if SleepingBeauty's age = 25 {
  SleepingBeauty, turnLeft.
}
else if SleepingBeauty's age = 30 {
  SleepingBeauty, turnRight.
}
else {
  SleepingBeauty, stayStill.
}
```
- For loop example:
```
repeat for number x is 5; x < 10; x is x+1 {
  say("ha").
}
```
- While loop example:
```
repeat until SleepingBeauty's age = 100 {
  SleepingBeauty, snore.
  age is age + 1.
}
```

Sample Programs

Sample Program #1 - GCD Function

```
Chapter GCD(number a; number b) return number {
  repeat until a = b {
    if (a > b)
      a is a - b.
    else
      b is b - a.
  }
  return a.
}
```

Sample Program #2 - 99 Bottles of Beer

```
Chapter Sing99BottlesOfBeer() {
  number bottles is 99;
  repeat until bottles = 0 {
    say bottles + " bottles of beer on the wall".
    say bottles + " bottles of beer".
    say "Take 1 down, pass it around".
    bottles is bottles - 1
    say bottles + " bottles of beer on the wall".
  }
}

Plot() {
  Sing99BottlesOfBeer.
}
```

Sample Program #3 - Creating A Monster

```
Character Monster {

  words trait name.
  number trait height.

  Action scare(words scream) {
    say scream.
  }

  Action crush(number personHeight) {
    if personHeight < height {
      crushed is true.
    } else {
      crushed is false.
    }
    return crushed.
  }
}

Plot() {
  Monster Monsta = new Monster(name is "Monsta"; height is "100").
  Monsta, scare("Be scared!").
  Monsta, crush(20).
}
```

Sample Program #4 - This Little Piggy

```
Character Foot {
  words list trait lyrics.
  Action pullBigToe {
    say lyrics[1].
  }
  Action pullLongToe {
    say lyrics[2].
  }
  Action pullMiddleToe {
    say lyrics[3].
  }
  Action pullRingToe {
    say lyrics[4].
  }
  Action pullPinkyToe {
    say lyrics[5].
  }
}

Plot() {
  Foot SingingFoot is new Foot(lyrics is ["This little piggy went to market"; 
"This little piggy stayed home"; "This little piggy had roast beef"; "And this 
little piggy went wee wee wee all the way home"]).
  SingingFoot, bigToePulled.
  SingingFoot, longToePulled.
  SingingFoot, middleToePulled.
  SingingFoot, ringToePulled.
  SingingFoot, pinkyToePulled.
}
```

Sample Program #5 - AliBaba and the Forty Thieves

```
~~Assuming Character types are defined in other files.
```

```
Plot() {
  Thief Captain is new Thief(id is 0).
  Thief list thieves is [].
  number thiefNumber is 1.
  ~~This creates a list of 39 thieves. Add the Captain and we have 40.
  repeat until thiefNumber is 40 {
    thieves, append(data is new Thief(id is thiefNumber; gold is 0; mood is
"unhappy")).
    thiefNumber is thiefNumber + 1.
  }

  Hero AliBaba is new Hero(work is "woodcutter"; gold is 1; nature is "good").
  SideCharacter Cassim is new SideCharacter(work is "merchant"; gold is 20;
nature is "malicious").

  Door SecretDoor is new Door(lockCode is "Open Sesame"; unlockCode is "Close
Sesame"; state is "locked"; gold is 1,000,000).

  Captain, tellDoor(SecretDoor; "Open Sesame").
  say(SecretDoor’s state) ~~unlocked

  repeat for number x is 1; x < 40; x is x+1 {
    ~~Increases gold and changes mood if the door is unlocked
    thieves[x], takeGold(SecretDoor; 1).
    say (thieves[x]’s mood).
  }

  Captain, tellDoor(SecretDoor; "Close Sesame").
  say(SecretDoor’s state) ~~locked

  AliBaba, tellDoor(SecretDoor; "Open Sesame").
  say(SecretDoor’s state) ~~unlocked

  AliBaba, takeGold(SecretDoor; 300).
  AliBaba, tellDoor(SecretDoor; "Close Sesame").
  AliBaba, say(whoamI).
  ~~Should print "I am AliBaba. My work is "woodcutter", my gold is 301, my
nature is "good".

  words list sillySimiles is ["barley"; "wheat"; "peanut"].

  Cassim, tellDoor(SecretDoor; sillySimiles, random).
  Cassim, getGold(SecretDoor; 100). ~~This should fail as door is still locked

  ~~To be continued...

}
```