Benjamin Morrison
Kaitlyn Evans
IF-Interpreter (Part 3)

If-Interpreter Class Design Reflection

From the first implementation of the project, we moved all of the objects that "drive" the program into our InterpretStory.h file. The main.cpp file is tasked with opening the if.html file, reading in the entire story, and reducing the story string to start at the first instance of a passage token and at the occurrence of the last passage token. This saves us time by not having to process through the other html information. This way once the string is passed into InterpretStory.h, it starts and ends with passages and nothing else. Once in InterpretStory, the program then parses through each passage to process the interactive story.

A function was added to the StoryTokenizer class, called passReturn. The constructor for StoryTokenizer will call passReturn. This constructor keeps track of the line numbers in which the passage beginning and passage ending tags occur and then pass these values into passReturn, which then returns the correct passage string according to the line numbers passed in. This string is then processed by InterpretStory.

- Syntax for locating name tag is located inside of InterpretStory.
- BlockTokenizer is separate class and no longer inside PassageTokenizer

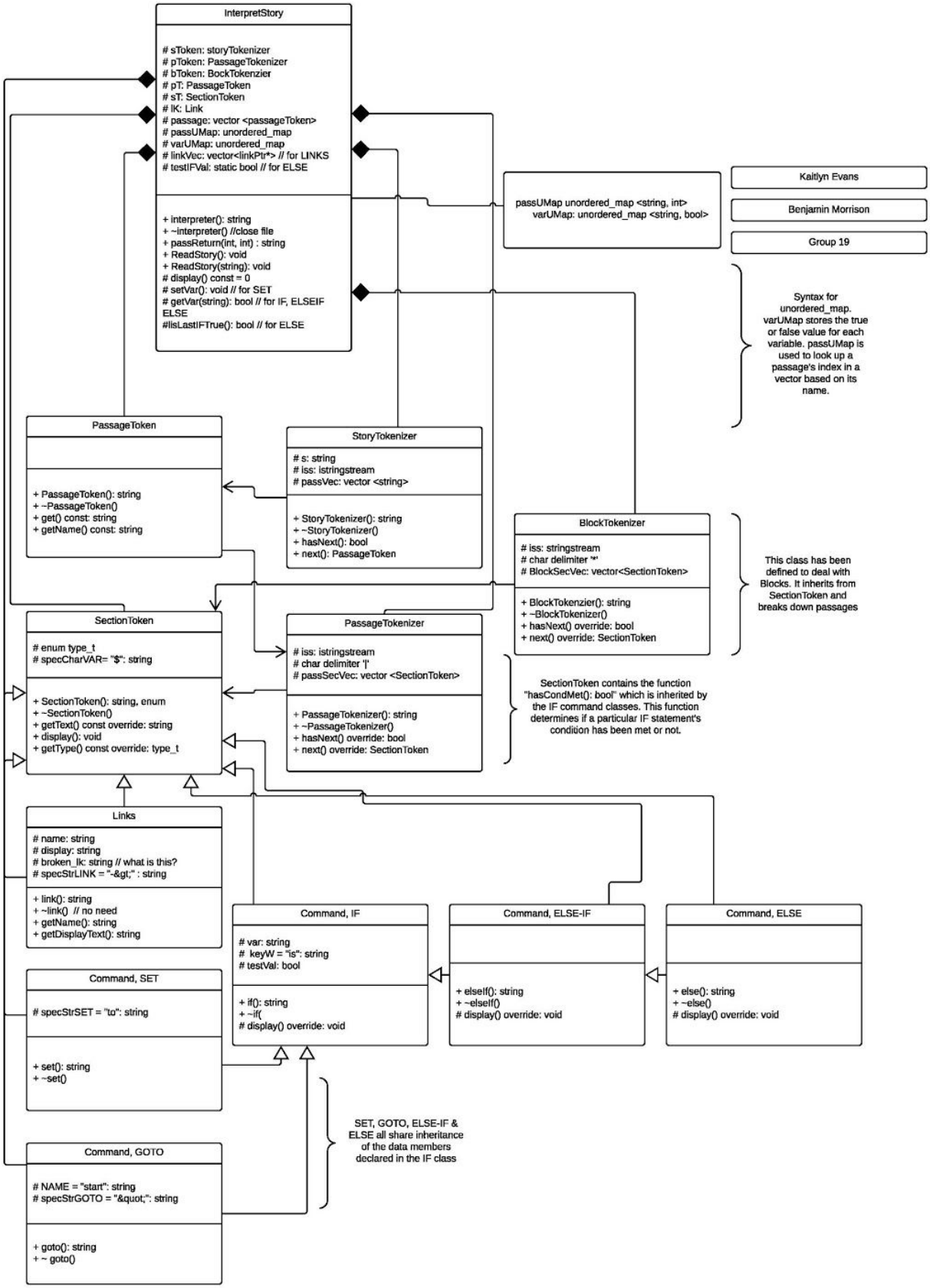
We took the main driver from Part 1 and filled our InterpretStory. Inside that main while loop we are filling our unordered_map in InterpretStory tasked while processing the story in conjunction. Thus, ideally, the unordered_map can be declared and filled with correct name and index information outside of the main loop so it is ready to be implemented. Before our fullStory goes to the next passage, we call our readPassage function. Inside readPassage function, we are recursively calling readPassage.

Additionally, we have classes for all the variety of commands. The CommandELSE-IF and CommandELSE inherit from CommandIF; and CommandIF inherits from SectionToken. Otherwise, the associations that were required for Part 1 are still apart of our Part 3. Furthermore, PassageToken, SectionToken, BlockTokenizer, BlockToken, CommandSET, and CommandGOTO are data members of our InterpretStory class. We also created a BlockTokenizer, and idealistically if deemed a block, we planned to create a BlockTokenizer blktok object -- this process will be very similar methods to how PassageTokenizer and PassageToken work. We could have just reused PassageTokenizer, however we decided that this way was better visualize the programs operations and they way it processes the story.

In our Part 2 we realized that we didn't exactly need to inherit as much as we had originally. So the change would now be as described above. Also we had to update the get functions, has functions, and hasNext functions. We didn't remember that with these particular functions, if they have different return values, we need to have different function names. Therefore we had to update all those, and get rid of the all the overrides. currState variable, tempMap data members from Part 2 were not needed. We added more helper functions inside the command classes that were not originally apart of Part 2. For example, getters, setters, and display functions.

Below is a UML diagram of our program before we began implementing major changes. After the UML diagram, we have included more changes we made as time went on and new challenges arose.

Cont…..


InterpretStory
sToken: storyTokenizer
pToken: PassageTokenizer
bToken: BockTokenzier
pT: PassageToken
sT: SectionToken
lK: Link
passage: vector <passageToken>
passUMap: unordered_map
varUMap: unordered_map
linkVec: vector<linkPtr*> // for LINKS
testIFVal: static bool // for ELSE
+ interpreter(): string
+ ~interpreter() //close file
+ passReturn(int, int) : string
+ ReadStory(): void
+ ReadStory(string): void
# display() const = 0
# setVar(): void // for SET
# getVar(string): bool // for IF, ELSEIF ELSE
#lisLastIFTrue(): bool // for ELSE
passUMap unordered_map <string, int>
varUMap: unordered_map <string, bool>
Kaitlyn Evans
Benjamin Morrison
Group 19
Syntax for unordered_map. varUMap stores the true or false value for each variable. passUMap is used to look up a passage's index in a vector based on its name.
PassageToken
+ PassageToken(): string
+ ~PassageToken()
+ get() const: string
+ getName() const: string
StoryTokenizer
s: string
iss: istringstream
passVec: vector <string>
+ StoryTokenizer(): string
+ ~StoryTokenizer()
+ hasNext(): bool
+ next(): PassageToken
BlockTokenizer
iss: stringstream
char delimiter '*'
BlockSecVec: vector<SectionToken>
+ BlockTokenzier(): string
+ ~BlockTokenizer()
+ hasNext() override: bool
+ next() override: SectionToken
This class has been defined to deal with Blocks. It inherits from SectionToken and breaks down passages
SectionToken
enum type_t
specCharVAR= "$": string
+ SectionToken(): string, enum
+ ~SectionToken()
+ getText() const override: string
+ display(): void
+ getType() const override: type_t
PassageTokenizer
iss: istringstream
char delimiter '|'
passSecVec: vector <SectionToken>
+ PassageTokenizer(): string
+ ~PassageTokenizer()
+ hasNext() override: bool
+ next() override: SectionToken
SectionToken contains the function "hasCondMet(): bool" which is inherited by the IF command classes. This function determines if a particular IF statement's condition has been met or not.
Links
name: string
display: string
broken_lk: string // what is this?
specStrLINK = "->" : string
+ link(): string
+ ~link() // no need
+ getName(): string
+ getDisplayText(): string
Command, IF
var: string
keyW = "is": string
testVal: bool
+ if(): string
+ ~if(
# display() override: void
Command, ELSE-IF
+ elseIf(): string
+ ~elseIf()
# display() override: void
Command, ELSE
+ else(): string
+ ~else()
# display() override: void
Command, SET
specStrSET = "to": string
+ set(): string
+ ~set()
SET, GOTO, ELSE-IF & ELSE all share inheritance of the data members declared in the IF class
Command, GOTO
NAME = "start": string
specStrGOTO = """: string
+ goto(): string
+ ~ goto()

As time carried on, we updated the inheritance in the Links class. Instead of inheriting from CommandIF, it inherits from SectionToken. Also, we have updated functions in CommandIF. Functions SetVar and SetTest find the instance of true and then sets the value to true, otherwise false, respectively. Also, GetVar and SetTestVal get the private data member and returns TestVal, respectively. The constructor for CommandIF only contains setVarAndsetTest.

ReadPassage implements a link vector in order to save all of the links. Two overload for readPassage, one takes in a passage token and the other takes in a block token. A switch statement for all the cases then implements the appropriate processes. A recursive implementation was also used for ReadPassage.

Lastly, we implemented the decision aspect of the interpreter for the user to choose the path they wish to take. This is in

- CommandSET now inherits from SectionToken.
- Syntax in CommandSET is the same as CommandIF, note it does not inherit from CommandIF. Additionally, comments inside classes will explain exact functionality.
- The Links class looks for “normal links” and “special links” which are denoted by a special character. From here we find and display the correct links accordingly. This is handled by two test cases.
- ComandELSE functionality will be handled in readPassage, class still exists.
- readPassage is outside of while loop in InterpretStory.
- CommandGOTO class finds the substring in between the special quotes. getName function was also implemented.
- BlockToken is structured like PassageToken. Brackets are located instead of “ < > ”, and creating a substring.
- Implemented the decision aspect of the interpreter for the user to choose the path they wish to take.