https://github.com/PeluzaVerde/FLCD

Documentation

My Symbol table uses one hash table with separate chaining. The hash table is a list of hashnodes, which are a singly linked list, each hashnode having a 'next' in the case of hashing to the same position. The symbol table saves the hashnode, key-value at the position indicated by the hashing of the key .The hash function I used is from the class Objects, a built in function, then modulo of the capacity of the hash table.

Hashnode-

K Key, V Value, Hashnode<K,V> next

K is the key of the node

V is the value of the node

Next is a pointer to the next node.

Hash table-

List of hashnodes, capacity, size.

The list of hashnodes are the buckets of the hash table, each is a singly linked list.

Capacity is the number of buckets of the hash table

Size is the current number of hashnodes.

Int Hash(Key K). Hashes the key to a position in the Symbol Table.

V add(Key K, Value V). Adds the key and value in the form of a hashnode. Returns the the old value if it exists, or null otherwise.

V get(Key K). Returns the value at the given Key.

Remove (K key). Removes a hashnode from the hash table.

Symbol table-

Hashtable of string and integer for key and value.

Integer addSym(String Key). Adds the symbol and returns the value. The key is actually the value getting hashed.

Remove(String Key). Removes a symbol from the symbol table.

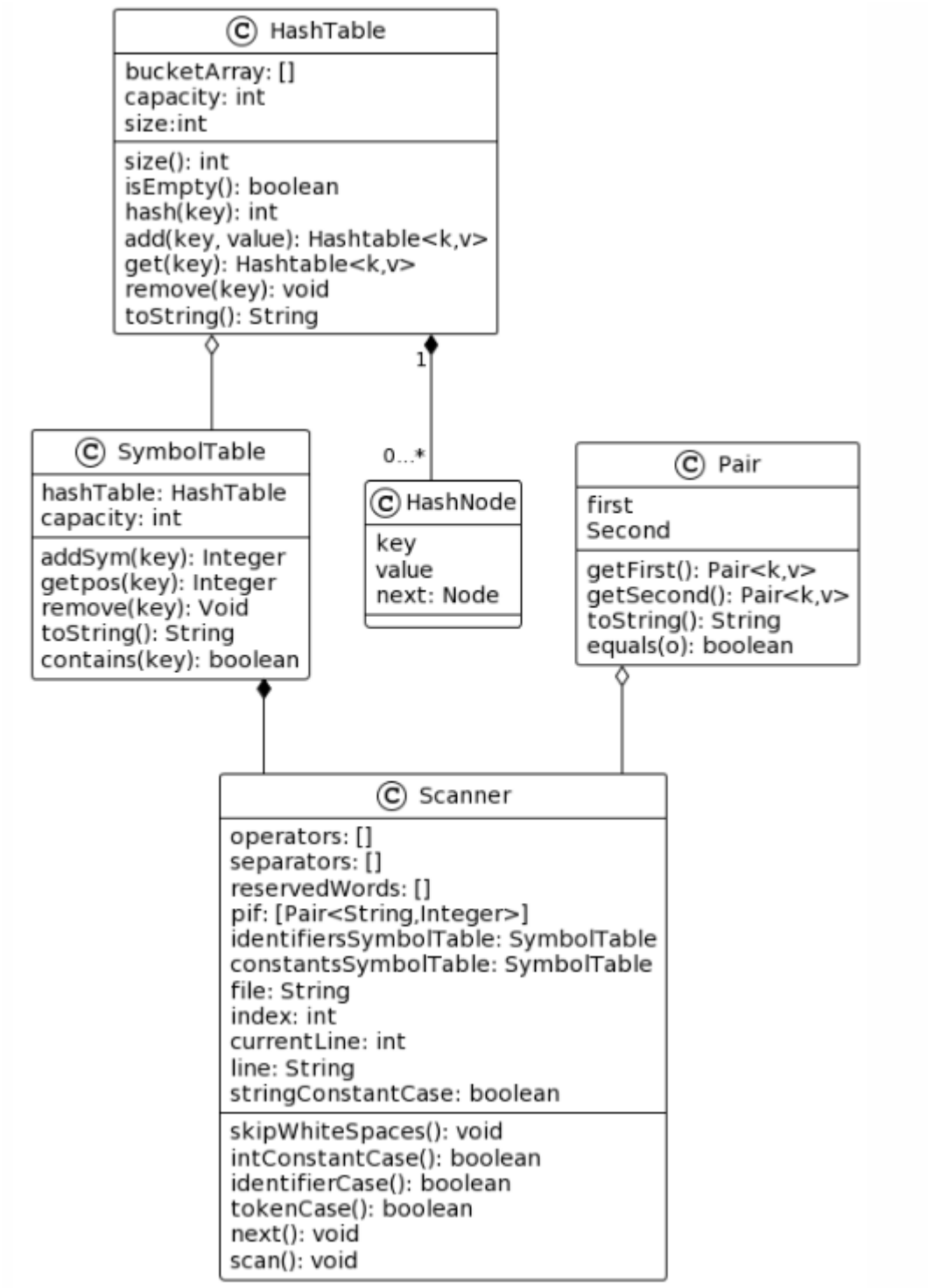
Int getPos(String key). returns the value of the key from the symbol table.

Bool contains(String key). Returns true if the symbol is in the Symbol table. False otherwise

Documentation for Scanner

1. Fields:
    1.1. Reserved words –initialized as an arraylist of all the reserved words
    1.2. Operators –initialized as an arraylist of all operators, including && and GE
    1.3. Separators –initialized as an arraylist of all separators
    1.4. `identifiersSymbolTable – a symbol table responsible for memorizing identifiers with their hashed code aka position in bucket array`
    1.5. `ConstantsSymbolTable – same as above, only that its used for constants only`
    1.6. Pif – a list of Pair of string and integer, string being the id/name aka ‘a’ ‘read’ ‘{’ and the integer the position in the symbol table its from. If it’s a token and not from a symbol table, its id is -1
    1.7. File – String that shows what program is getting scanned. Parameter used to find the file
    1.8. Index – used to go thru the line character by character, or by the length of the reserved word/operator
    1.9. CurrentLine – The number of the line the scanner is currently at. Used for letting the user know at which line there is a lexical error
    1.10. Line –The line is read in the scan function, being a line of code. After its done parsing thru it, it goes to the next line and resets the index.
2. Functions:
    2.1. SkipWhiteSpaces() -Skips white spaces and advances the 'index' within the 'line' string
    2.2. StringConstantCase()- Using Regex it makes sure it is a proper String. If it’s a match to the regex(which is how all cases work) it adds it to the constants symbol table, adds it to the Pif using itself(the string) and its position from the symbol table, then returns true. If It doesn’t match this returns false
    2.3. IntConstantscase()- makes sure the expression is an int ie –4 or 543, then it checks if its an invalid integer (453MISTAKE5544). Adds it to pif like in the above function
    2.4. IdentifierCase()- finds an identifier, which must start with a letter. IF this is in the reserved words list it returns false. if it already exists in the symbol table, then it just adds it to the pif with the same position index Otherwise it adds it to the IDENTIFIERS table and to pif.
    2.5. Tokencase() - using a possibletoken string, it checks if it matches with operators, separators and reserved words, making sure to check for cases when there is a compound operator (ie GE, &&, = and ==)
    2.6. Next() - we skip white spaces, if we are at the end of the line we also return. we check for the above 5 cases. If neither returns true, then it is a lexical error, throwing an error at the location of it.
    2.7. Scan()- reading from a file, we parse the program , and for each line of code we call ‘next’. After all the code was parsed, we print to two different files the symbol table contents and the PIF.

Pair
class:
a pair of variables (typically used in scanner to hold the key/constant and its position
first and second.
Has methods to get first and get second.
Boolean equals(Object o): checks if the object and an instance of pair are equals(same class, same same first value, same seconds value)


HashTable
bucketArray: []
capacity: int
size:int
size(): int
isEmpty(): boolean
hash(key): int
add(key, value): Hashtable<k,v>
get(key): Hashtable<k,v>
remove(key): void
toString(): String
1
0...*
SymbolTable
hashTable: HashTable
capacity: int
addSym(key): Integer
getpos(key): Integer
remove(key): Void
toString(): String
contains(key): boolean
HashNode
key
value
next: Node
Pair
first
Second
getFirst(): Pair<k,v>
getSecond(): Pair<k,v>
toString(): String
equals(o): boolean
Scanner
operators: []
separators: []
reservedWords: []
pif: [Pair<String,Integer>]
identifiersSymbolTable: SymbolTable
constantsSymbolTable: SymbolTable
file: String
index: int
currentLine: int
line: String
stringConstantCase: boolean
skipWhiteSpaces(): void
intConstantCase(): boolean
identifierCase(): boolean
tokenCase(): boolean
next(): void
scan(): void