GitHub: [https://github.com/Phe0niX12/FTCD](https://github.com/Phe0niX12/FTCD/tree/main/LAB%202/src)

DOCUMENTATION

Symobol Table

I implemented a Symbol Table which is composed of 3 Hash Tables, one for identifiers, one for string constants and one for integer constants. I resolved the collisions issue by using a list for each position of the hash and adding the elements with the same hash code to the list one after the other. The Hash Table is generic.

OPERATIONS:

HashTable:

getCapacity()=> returns the capacity of the hash table

Worse: O(1)

Average: O(1)

Best: O(1)

hash(int key)=> returns the hash code of the key for an int key

Worse: O(1)

Average: O(1)

Best: O(1)

hash(string key)=> returns the hash code of the key for a string key by adding the ASCII codes of the letters of the word

Worse: O(n)

Average: O(n)

Best: O(1)

getHashValue(T key)=>return the hash position of any type of key by using the other 2 hash functions

Best: O(1)

Average: O(n)

Worse: O(n)

contains(T key)=> returns true if the key is in the hash table and false otherwise

Best: O(1)

Average: O(n)

Worse: O(n)

getPosition(T key) => returns the positon of the given key in the hash

Best: O(1)

Average: O(1)

Worse: O(1)

add(T key) => adds the key to the hash table and return it’s position

Best: O(1)

Average: O(n)

Worse: O(n)

Symbol Table:

addIdentifier(String name) => adds the identifier to the identifier Table

Best: O(1)

Average: O(n)

Worse: O(n)

addIntConst(int constant) => adds the int constant to the int constant Table

Best: O(1)

Average: O(1)

Worse: O(1)

addStringConst(string constant) => adds the string constant to the string constant Table

Best: O(1)

Average: O(n)

Worse: O(n)

getIntConstPos(int const) =>returns the position of an int constant from the int constant Table

Best: O(1)

Average: O(n)

Worse: O(n)

getStringConstPos(string const) => returns the position of a string constant from the string constant Table

Best: O(1)

Average: O(n)

Worse: O(n)

getIdentifierPos(String name) => returns the position of an identifier form the identifier Table

Best: O(1)

Average: O(n)

Worse: O(n)

hasIdentifier(string name) => returns true if there is the identifier in the Identifier Table

Best: O(1)

Average: O(n)

Worse: O(n)

hasConstantInt(string name) => returns true if there is an int constant in the Int Constant Table

Best: O(1)

Average: O(n)

Worse: O(n)

hasConstantString(string name) => returns true if there is the string constant in the String Constant Table

Best: O(1)

Average: O(n)

Worse: O(n)

Scanner:

The Scanner class is responsible for the build of the program internal form and the symbol table. It has to check if the input program is lexically correct or not. The PIF(Program Internal Form) is a list composed from pairs of the form string and the position in the Symbol Table(Position which is composed of a pair of integers for evading collisions in the hash). The string from the PIF is the token, string const, int const, identifier. We consider for the tokens the position in the symbol table to be (-1,-1).

Operations:

setProgram(string program) => sets the program on which the build is made

readTokens() => reads from the token.in file the tokens and separates them in reserved words and other tokens

skipSpaces() => skips the spaces from the given program in order to make the program readable

caseStringConstant() => checks if a string is lexically correct and if it is it adds it to the Symbol Table if it doesn’t already exist there and to the PIF and then updates the index and returns true, if not it returns false

caseIntConstant() => checks if a int is lexically correct and if it is it adds it to the Symbol Table if it doesn’t already exist there and to the PIF and then updates the index and returns true, if not it returns false

caseIdentifier() => checks if we have a valid identifier in the program( strats with \_ or a letter, contains only letter, digits and \_) and if in the PIF we have declared an token of identification and if all those are respected it adds to the symbol table if it does not exist and to the PIF; if the conditions are not respected it returns false

checkIfValid() => checks if a possible identifier is valid by changing if an identification token is present in PIF before the current expression or if it is present in the Symbol Table, if neither it returns false, else it returns true

nextToke() => treats the current case and if there is no corresponding case throws an exception error

scan(string program) =>reads the program from the given file, it writes the PIF and the Symbol Table to 2 different files after if checks for token to the very end of the program. It also displays a message if the program is lexically correct; else it displays an error message.