Documentation – Scanner

Requirements:

Based on the specification of the mini-language chosen in the first part of the lab 1 implement a scanner that will take as input a text file containing the source program and will produce as output the following:

* PIF -> Program Internal Form
* ST -> Symbol Table

In addition, the program should be able to determine the lexical errors, specifying the location, and if possible, the type of the error.

Some additional restrictions:

1. Identifiers:

b. arbitrary length, no more than 250 characters

2. Symbol Table:

b. separate tables for indentifiers, respectively constants

3. Symbol Table Organization:

c. hashing table

The scanner is implemented in the C++ language.

The codification table is kept in memory as a map<string, int>.

The PIF is a vector<pair<int, int>>

There are two separate symbol tables, one for identifiers and one for constants, both implemented as hash tables on maps (map<int, string>). The collision strategy is open adressing, and rehashing when the tables are full.

First the file containing the program is read line by line. Then, the tokenizing is done using a regex.

As the tokens of each line are processed, the symbol tables for identifiers and constants are built. The functions isConstant and isIdentifier are used to differentiate between the two types.

bool isIdentifier(string const &token) {  
 return (regex\_match(token, regex("([A-Za-z][A-Za-z0-9]\*)")) && token.length() < 250);  
}  
  
bool isConstant(string const &token) {  
 return regex\_match(token, regex(R"(0|(\+|-)?[1-9][0-9]\*|'[0-9A-Za-z]')"));  
}

All the tokens constants are also kept in a temporary variable. After the symbol table construction is finished, the pif is built from the list of tokens.