Christopher Diehl & Sandeep C Mattappali
Tokenizer

GENERAL OUTLINE OF PROGRAM :
In the tokenizer struct a copy the input string is kept and each time TKGetNextToken is called , we move along this string using a counter called _processedLen which keeps track of how much the string we have processed.

A switch statement is used to handle the transition from one state to another. This is used to distinguish between the basic types like word , hex , octal , float and int.
In the tokenizer struct there is a field called _state which hold the STATE of the transition . The STATE is created using an enum, which holds the various states like int , octalc_operator etc.
In order to handle the c operators a much more complex approach is used. If the state of the program is in start then it checks if the character, p, is a special character (any c op longer than 2 chars, like >>= or sizeof(). If it is a special c op character it returns the character while incrementing the character pointer and the processed length accordingly. Later on, the program checks for a normal c op character in the stateAndCharTest method to find any c_op less than length 2. If the character is a punctuation character but not a c operator then it returns 0 which means p is a bad token, this was used to handle cases such as $ or @. The bad token or c operator is then returned to main and freed.

If the token state has changed from Start to something besides bad token the program keeps looping through the copy of the input string, if it equals bad token a token is returned. We create a token using our own version of strncpy. The reason we created this version is because we wanted the program to be a little more knowledgeable and not go past the memory location of the string to be copied. We then return the token and free the token in main if the token is not equal to null. If at any time the token returned to main is equal to null, the program quits.

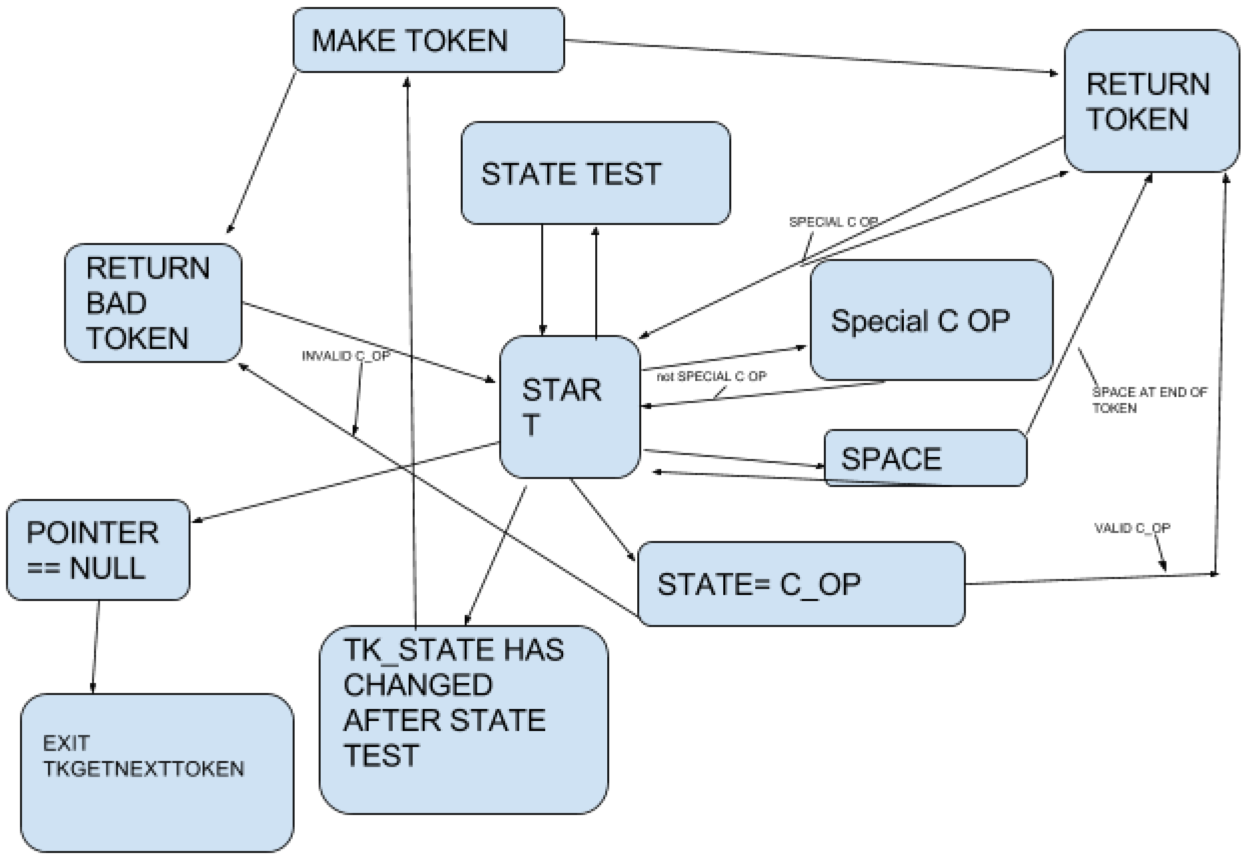
In order to get the type of the program, we look up the state of the token in the tokenizer struct, except for c operators. For C operators we find the token in a tokenArray, then acquire the corresponding tokenType. The reason we used two arrays for C Operators is because we wanted to make it as easy as possible to expand in the future, and we felt that arrays would be easier to expand and edit than a gigantic switch statement.

Some tradeoffs we had to make:
We wanted to try to handle the special c operators like sizeof() as elegantly as possible while making it as easy for us to debug and write, so we decided to malloc and copy the length of the special c_operator token to a sample token, and if the sample token matched the predefined char * exampleCOPHere rather than iterating through the inputString to see if the inputString had any specialCharacters. We thought that using predefined methods such as __strncpy and

strcmp would look more clean and allow us to more easily accommodate more special c operators. See checkForSpecialC_OP function.

In order to handle the c comment /*... */ the program increments the pointer to determine if the pointer *p ==’*’ and *p++ =’/’. However if *++p !=’/’ then it decrements the pointer by one. For readability between partners, in an attempt to make the program more clear, I incremented the everything by one, and then decremented everything by one if *++p !=’/’ in order to convey that everything needs to be incremented, updated for every char in the comment. See processComment function.

STATE MACHINE DIAGRAM



AMBIGUOUS CASES :

1) decimal followed by hex

   EG : " 1231230X12321313 "

   This is considered as a decimal followed by a word

   decimal integer "1231230"

word "X12321313"

2) single zeros and mutiple zeros :

0 -> decimal integer
00 -> octal integer
000 -> octal integer
000* -> octal integer      // kleene star

3) float followed by hex

EG : " 123.213e-1230x12323 "
This is considered as a float followed by a word
float "123.213e-1230"
word "x12323"

4)Bad tokens:

0x
1.
1.1e
#
?
@
*                                  // Muliple or single back slashes

Extra Credit :

1) C Comments :
    i) Both types of C comments are handled , but in handling /* **/ style tokens, if there are only the c comment in the string , then the program outputs nothing
2) C KeyWords :
    i) All 32 C keywords are handled , but only the lower case version , like a real C Compiler. Meaning WHILE is not recognized , but while is .
    More over if there are no spaces between the keyword the c keywords are not recognized

        eg : whileiffor is not handled but
             while if for recognizes the 3 keywords

    Implementation :

        we first recognize that that keyword is a word type . Then when we try to print the word , we test whether the word is a c key word and if it is then
3) Single Quote Double Quote
    i) Not handled as this will lead to drastic changes in the program. The program only takes a single string as input .