GetToken

Part 1: Rewriting Cstring Functions

Project Description:

The set of functions written in this portion of the project will be lower-level functions used to work with character arrays. **None of the functions can use built-in cstring functions. Everything must be rewritten using character arrays.

Lower-Level Cstring Functions:

```
void StrCpy(char MyString[], char CopyThis[]); //CopyThis into MyString
void StrCat(char MyString[], char CatThis[]); // CatThis to the end of MyString
int StrLen(char MyString[]); //return length of MyString
int StrCmp(char MyString[], char Compare[]); //compare two strings
void SubStr(char MyString[], char Sub[], int start, int length);
//create substring of MyString with a certain length and starting position
int Find(char MyString[], char FindThis[], int pos);
//return location of FindThis in MyString
int Find(char FindHere[], char target, int pos);
//find a target character starting at a particular position
int FindAny(char MyString[], char FindThese[], int pos);
//return position of first occurence of anything in MyString, in FindThese
int FindNotAny(char MyString[], char FindThese[], int pos);
//position of first occurence of anything in MyString, not in FindThese
```

Additional functions to include:

```
int ToInt(char MyString[], int pos);
//return first int in MyString starting from a particular position
int AtoI(char ConvertThis[]); //convert array of characters to an integer
int CharToInt(char ch); //convert a single character to an integer
void ToLower(char MyString[]); //makes every character in an array lowercase
bool Eq(char MyString[], char Compare[]); //true if MyString=Compare
bool GT(char MyString[], char Compare[]); //true if MyString>Compare
bool LT(char MyString[], char Compare[]); //true if MyString<Compare
```

Purpose:

The purpose of rewriting cstring functions is to gain a better understanding of cstrings, character arrays, and the associated cstring functions.

Notes:

Additional function descriptions:

- `void StrCpy(char MyString[], char CopyThis[]);`
  All the characters in CopyThis will be copied into MyString, along with a NULL character at the end.
  After copying, the length of MyString should be equal to the length of CopyThis.

- `int StrCmp(char MyString[], char Compare);`
  Corresponding characters of each array will be comapred using their ASCII values to determine which string is "larger". If the first string is larger, return 1. If the second string is larger, return -1.
  If the two strings are the same, return 0.

- `void SubStr(char MyString[], char Sub[], int start, int length);`
  Starting from the "start" index of MyString, characters will be copied into the substring. "Length" specifies how many characters will be copied over.

- `int Find(char MyString[], char FindThis[], int pos);`
  This Find function is an overloaded function for which the second parameter is a character array. The function must find the entire string inside MyString and return the index number corresponding to where the first character of the FindThis string is located inside MyString.

- `int FindAny(char MyString[], char FindThese[], int pos);`
  FindThese is the set of characters to be searched for inside MyString. This function returns the index number corresponding to where one of the characters inside FindThese was found inside MyString.
  Possible bug Do not return the index number that corresponds to the character inside FindThese.
  For example, FindAny("Super", "aeiou", 0) will search "Super" for the first occurrence of any character in the set "aeiou" starting from the 0th index of "Super". The first occurrence is the 2nd letter of "Super", which is the 5th letter in the character set. The function should return 2, not 5.

- `int FindNotAny(char MyString[], char FindThese[], int pos);`
  In the same way FindAny finds the first character in MyString that is inside the set FindThese, this function finds the first character in MyString that is NOT found inside the set FindThese, and returns the index number.

Part 2: Tokenizing a Block of Text

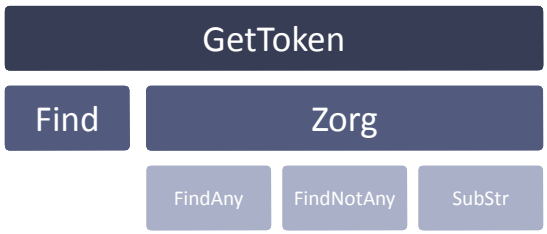
Project Description:

The second portion of this project is to write a program that will extract tokens from a block of text. There will be two new necessary functions (Zorg and GetToken) that will utilize the lower-level functions from Part 1. **The two functions must be built entirely using the previous functions. This means that there should not be any loops inside these functions.

```
int Zorg(char MyString[], char CharSet[], char Substring[], int pos);
//Creates a substring of characters from MyString that can be found inside a character
set, starting from a particular position.

bool GetToken(char Block[], char Token[], char &tokenType, int &pos);
//Determines what type of token a character represents, and creates an array
containing the token.
```

All the tasks that GetToken needs to perform can be broken down to smaller tasks handled by other functions.



GetToken should only rely on the Find function from Part 1 and the new Zorg function.

This is a general outline of how GetToken will rely on a hierarchy of functions to do its task. A more detailed description of how the task is broken down is written out under the "Notes" section.

The final part of this project is to extract tokens from a file, create a sorted two-dimensional array of all the tokens (excluding repeats), a corresponding integer table of the number of times each token occurs inside the text files, and display the two arrays as a table.

Purpose:

The ultimate purpose of this project is to implement top-down design. Lower-level functions should be written so that they perform a very simple task which can be re-used for many purposes. If the functions are versatile enough, the task of writing a higher level function such as GetToken becomes very simple and intuitive.

Notes:

Layout of GetToken:

Establish 3 character array sets for each token type: Alpha, Num, Punc

Block[] = "742 Evergreen tr."

`GetToken(Block, Token, type, 0)` will check position 0 of Block.

Block[0] is the character '7'

`Find(Alpha, '7', 0)` will determine whether this character is a letter of the alphabet.

Call Find for the 3 character sets until it is found in one of them.

The character will be found in the Num array, so the tokenType is a number.

`Zorg(Block, Num, Token, 0)` will create a substring of numbers and save them into the Token array.

`FindAny(Block, Num, 0)` will return the index of the first character that is a number.

The first character starting from position 0 that can be found in Num is '7' at position 0.

`FindNotAny(Block, Num, 0)` will return the index of the first character that is NOT a number.

The first non-number character starting from position 0 is a space at position 3.

The token spans positions 0-3 in Block, and has a length of 3 characters.

`Substring(Block, Token, 0, 3)` will create a substring of 3 characters starting from position 0.

Token becomes "742". ** Include the NULL character**

Push the position over to the next token.

The new value of position becomes 3.

If there is another token in the new position, return true. If position gets moved over to the end of Block, return false to express that there are no more tokens to process.

`GetToken(Block, Token, type, 3)` will check position 3 of Block.

The character "E" will be found in the Alpha set. The tokenType is a word.

Zorg will create a substring containing "Evergreen".

The process is repeated until the final token '.' has been processed. There are NO LOOPS inside GetToken.