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 StrStr(char myString[], char findThis[], int pos);
//return location of findThis in MyString
int StrChr(char findHere[], char target, int pos);
//find a target character starting at a particular position
int FindAny(char myString[], char charSet[], int pos);
//return position of first occurence of anything in charSet, in myString
int FindNotAny(char myString[], char charSet[], int pos);
//position of first occurence of anything in myString, not in charSet
```

Additional functions to include:

```
int ToInt(char myString[], int pos=0);
//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.

- `int FindAny(char myString[], char charSet[], int pos);`
  charSet is the set of characters to be searched for inside myString. This function returns the index number corresponding to where one of the characters from charSet was found inside MyString.
  Possible bug Do not return the index number that corresponds to the character inside charSet.
  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 charSet[], int pos);`
  In the same way FindAny finds the first character in MyString that is inside the set charSet, this function finds the first character in myString that is NOT found inside the set charSet, and returns the index number.

Part 2: Tokenizing a Block of Text

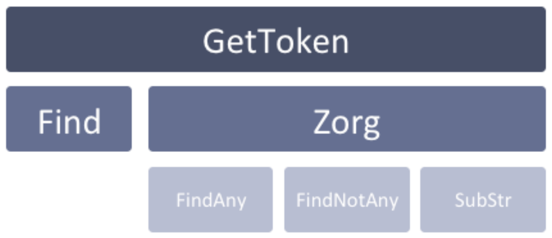
Project Description:

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 – horrible name, I know. Give me a suggestion and I'll change it!-- 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 token[], int pos);
//Creates a substring of characters from myString that can be found inside a character
set, starting from a particular position, copies it into token.
//returns true if it successfully copied the token. Return false if end of string is
reached without copying the token.

bool GetToken(char block[], char token[], char &tokenType, int &pos);
//Grabs the next token starting from position pos in block and copies it into token.
tokenType is the type of this token.
//returns true if it successfully copied the token. Return false if end of string is
reached without copying the token. This is essentially the return value of the Zorg
function.
```

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 StrChr (labeled Find in the image) 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.

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:

- Examin the current character block[pos]. decide what type of character this is (see if it can be found in the ALPHA, NUMBERS, PUNCTUATION, or SPACES character sets. Copy the appropriate character set into the variable charSet:
    - `StrCpy(charSet, ALPHA);` for example.
- Set the appropriate value of tokenType
- Call Zorg with the block, current position, and this character set.
- Return the return value of Zorg

You can test the GetToken function by slicing a string of mixed tokens into individual tokens and printing them:
```
char s[] = “123 Elm Street. Pasadena, CA 91106. USA. Email: tokenizer@token.com”;
int pos = 0;
char token[MAX_TOKEN_SIZE”];
int type;
while (GetToken(s, token, type, pos)){ //Stop when end of string is reached.
       cout<<token<<”  “<<pos<<endl;
}
```

Part 3: Tokenizing a Block of Text

The final part of this project is to extract tokens from a file.

Save a large text file into your project folder.

Read blocks of 10,000 characters at a time into a block of chars using the read( ) function

Pass the block of chars to GetToken in a loop to tokenize the string.

…

Create a report with this information:

1. Number of words in the file.
2. Number of numbers in the file.
3. Number of punctuation in the file.
4. List and frequency of the top twenty most frequent words in the file.

GetToken – Overloaded function:
```
bool GetToken(istream& inFile, char block[], char token[], char &tokenType, int &pos);
//Grabs blocks of text from the inFile, copies them to the block[] array and copies
the (next) token into the token[] array. On future calls to this function, the next
tokens will be copied into the token array[]…
//Once the end of the block[] string has been reached, another block is retrieved from
the file and copied into block[] and this process continues.
//once the end of the file has been reached, false will be returned, otherwise (in
case of a successful token retrieval), true is returned.
Ifstream ins;
Ins.open(“myfile.txt”);
//Test for fail here…

while(GetToken(ins, sentence, token, type, pos)){
       Process(token, type);
       //process the token here.
}
Ins.close();
```