Technical Documentation:

This scanner is designed to take in a file written in the PL language and output a separate file of sequential tokens discovered in the original file. The way this scanner accomplishes this is by only reading forward until it sees a white space. The string that is read in is then evaluated to be a Token. That Token is output to the new file.

1. Purpose of the software is to create an intermediate code that is readable by the parser, or the next stage of compiler. As such we grouped Tokens together to be used by the parser, and removed any white space or comments in the original program.
2. Design:
    1. Classes:
        1. **Token**: The token is the most basic class to analyze. It has two member variables, and just a handful of functions.
            Member variables:
                sname: this is of symbol type and denotes the type of token
                svalue: This is actually a struct of attval. The attvalue displays the attributes associated with the token, the integer value the token holds and the lexeme of the token
            Functions:
                Token(): Default constructor – sets to sname to NONAME; sets the svalue.value to -1 and the svalue.lexeme to the empty string
                ~Token(): Default Deconstructor – isn't used yet
                Token& operator=(const Token& t): overloaded copy operator assigns the member variables of the right side to the left side
                insert(ostream &os): formats the way Token would be ouptuted for the outputstream
                GETTERS:
                getSymbol(): returns the sname
                getValue(): returns the integer value in struct attval
                getLexeme(): returns the string lexeme in struct attval
        2. **SymTable:** The symbol table class is a class around an array of tokens. This array stores the different reserved words in the language as well as any new identifiers that the user creates in PL.
            Member Variable:
                occupied: an integer that stores the running total of all indexes filled
                vector<Token> htable: The hashtable that holds all the values. Created as a vector of type Token in order to store more information
            Global Variable:
                SYMTABLESIZE: defined here as 307 we could change it if desired
            Member Function:
                hashfn(string s): The hash function is how we calculate the index for a string. The parameter s computes what index the string will belong to. Only called by insert() or search()
            Functions:
                Symtable(): Default constructor, sets occupied to 0 and creates the initial vector to be of size SYMTABLESIZE

void loadResvd(): Loads the predefined reserved words into the htable. Implementation uses bad practice magic numbers. A more adequate solution may be added given enough time

~Symtable(): Default destructor

int search(string s): Uses hashfn to locate the index where a string s exists. The string will be the lexeme of the Token at that location

int insert(string s, Token &t): Uses hashfn() to locate the index where the token is inserted and place token t into that spot. As of now there is no case for collisions which will need to change in future iterations

const Token& index(int I): returns a reference to the token situated at index I. This function was created to ease my suffering but I haven't use it yet. Will remove it for coverage stats if necessary

bool full():Simple boolean returns true onle if the occupied variable is equivalent to SYMTABLESIZE

int getOC(): returns the occupied variable

void printTable(): An error checking function, prints out the whole htable to the command line NOT THE OUTPUTFILEwill likely comment out for coverage stats

3. Scanner: The scanner is the heavy lifter of this program. The primary role being getToken() which will get called by administration and determines which Tokens to assign to the input.

Member Variables:

ifstream *inputfileptr: of type ifstream this pointer is passed in with the constructor of the scanner and will point to the source material. This pointer will be used to read new characters to be scanned

symtable *symtableptr: of type symtable this pointer is passed in with the constructor as well. This pointer is used to communicate with the passed in symbol table, be it checking for existing Tokens or passing new symbols into the table

char laChar: arguably the most important variable in this class, it always one character ahead of the character being acted on. In this way the scanner will know what is coming and be able to act appropriately

Member Functions:

bool isXX(char achar): all of the bools accomplish roughly the same thing just specialized for their situation. Their function names are descriptive for a reason

Token recognizeXX(): These functions are what actually create tokens out of the input stream. They are called by getToken() once it determines what kind of Token to create. The recognize functions specify the different Tokens to be created whether they be numbers, special characters or names. With Names including both keywords and identifiers.

Token recognizeComment(): Slightly different from the other recognizes, it strips away the entire line when it sees a comment and then returns back to getToken to slot into another recognize function

Functions:

Scanner(): default scanner has nothing in it. Can’t do much without an input stream which is passed in the implicit constructor

Scanner(ifstream &instream, Symtable &symboltable): implicit constructor. Uses inline function to store the values passed by reference into the member pointers
~Scanner(): default destructor, unused
Token getToken(): The function that does all the work, it analyzes the current character as well as the lookahead character to determine what kind of Token is going to be needed. The 3 recognize functions correspond to significant patterns in the first two characters

4. Administration: covers all the rest of the utilities needed that doesn't quite fit into the other classes.
    Member variables:
        ifstream *inputfileptr: of type ifstream this pointer will point to the input file and is passed in by reference in the constructor
        ofstream *outputfileptr: of type ofstream this pointer will point to the output file and is passed in by reference in the constructor
        Scanner &scanr: of type Scanner this will be able to call the scanner and its functions and in relation to the actual files
        int lineNo: an integer that holds the current line number for error reporting purposes
        bool correctline: is set to true only if the line was all correct is set to false if an error is encountered
        int errorCount: an integer that tracks the total amount of errors
    Functions:
        Administration(ifstream& in, ofstream &out, Scanner &sc): implicit constructor, variables are assigned to their respective pointers
        ~Administration(): Default constructor, unused yet
        void NewLine(): incrases the line number and resets correctline to true
        void error(string text): an error reporting function
        int scan(): the driver of the Scanner variables

2. How they work together:
    Symtable is made of Tokens. Scanner creates Token out of the input and stores new IDs in Symtable. Admin manages the files and the scanner
3. Limitations:
    This was a lot of work to accomplish with one brain, the functions currently don't work together as expected all the time, so limitations are time and manpower. Some extensions would be to output the tokens properly into the output file as well as proper error reporting