CS117 (Fall 2019) Programming Assignment 1

Given code (attached C++ code) is an interpreter of a language named Simplified-Infix-Expression that we discussed in class. The original grammar (left-recursive) is:

E → E + T | E - T | T
T → T * F | T / F | F
F → Num
Num → 0 | 1 | 2 | 3 | . . . | 9

After converting the grammar to a corresponding right-recursive form, the resulting grammar is:

E → T E2
E2 → + T E2 | - T E2 | ε
T → F T2
T2 → * F T2 | / F T2 | ε
F → Num
Num → 0 | 1 | 2 | 3 | . . . | 9

Now, we can write an interpreter for this language with the tail-recursion method. The given code is an implementation using the recursive descent parsing technique in which each non_terminal is implemented with a function.
Note that the input expression allows only single digit numbers and no spaces are allowed.

Your missions (within the Lab session)

(1) Compile and run the interpreter using a sample program to this language (Simplified-Infix-Expression), which does not allow spaces.

Please try the following expression and check the result:

2+3*4/2+3+4*2

(2) Insert a trace code (a simple cout statement with function name) in each function (beginning part) and display the sequence of function names executed during the run.

(3) Upgrade the code for the following requirements:
1. Input is from a text file;
2. input expression can include spaces;

After completion of the code, please create a text file and type: 2 + 3 * 4 /2+3 + 4 * 2
(be sure to include spaces) and run the program with the file.
If your interpreter does not return the correct result, correct your code until well done.

(4) Decorate your code with a good documentation, i.e., the global documentation and each function head documentation.

The global documentation should include the description of the whole program (what it does, how it is implemented, etc.), input/output description, how to compile/run, etc.
Each function head doc. should describe what that function does, input/output, etc.

Before leave, you should do the demo to the instructor.

```
//// Interpreter for simplified infix expression with {+, -, *, / } operations;
//// Keyboard input, single digit numbers only and no spaces are allowed;
//// compile:  $>g++ prog1.cpp
//// run with:  >2+3*4/2+3+4*2

#include <cstdlib> //for atoi()
#include <iostream>
using namespace std;

int Exp(), Term(), Exp2(int), Term2(int), Fact();
string prog;      //string for reading 1-line input expression (program)
int indexx = 0; //global index for program string

int main(int argc, const char **argv)
{ cout<<">";
  cin>>prog;
  cout<<"result= "<<Exp()<<endl;
}

int Exp()
{ return Exp2(Term());
}
int Term()
{ return Term2(Fact());
}

int Exp2(int inp)
{ int result = inp;
  if (indexx < prog.length())     //if not the end of program string
  { char a = prog.at(indexx++); //get one chr from program string
    if (a == '+')
       result = Exp2(result + Term());  //handles T+T
    else if (a == '-')
      result = Exp2(result - Term());  //handles T-T
  }
 return result;
}

int Term2(int inp)
{ int result = inp;
  if (indexx < prog.length())     //if not the end of program string
  { char a = prog.at(indexx++); //get one chr from program string
    if (a == '*')
       result = Term2(result * Fact()); //handles consecutive * operators
    else if (a == '/')
       result = Term2(result / Fact());  //handles consecutive / operators
    else if (a == '+' || a == '-')   //if + or -, get back one position
       indexx--;
  }
 return result;
}

int Fact()
{ char a = prog.at(indexx++);  //get one chr from program string
  return atoi(&a);  //converts a char to a numeric number and return
}
```