Prog. #2 _ 1082414

如何執行:

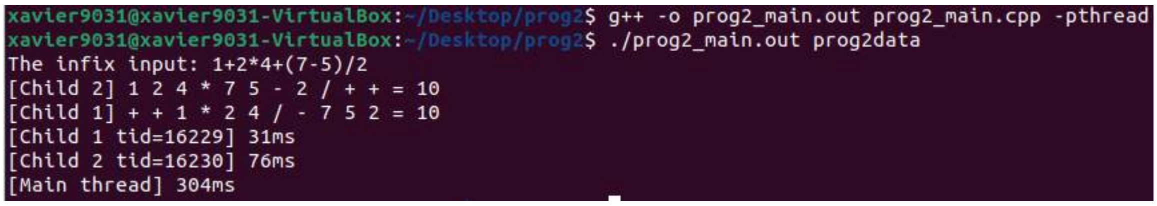
Compile:

```
$  g++ -o prog2_main.out prog2_main.cpp -pthread
```

Execute:

```
$  ./prog2_main.out prog2data
```

Example:



```
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ g++ -o prog2_main.out prog2_main.cpp -pthread
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ ./prog2_main.out prog2data
The infix input: 1+2*4+(7-5)/2
[Child 2] 1 2 4 * 7 5 - 2 / + + = 10
[Child 1] + + 1 * 2 4 / - 7 5 2 = 10
[Child 1 tid=16229] 31ms
[Child 2 tid=16230] 76ms
[Main thread] 304ms
```

設計理念:

一開始執行後先用start來存Main thread開始執行的時間點，讀入檔案，輸出檔案中的內容，接著透過Reader將Operator跟Operand區分開來並存在一個由自訂的資料型態Token組成的vector中，再自訂一個my_data資料型態來儲存需要交換用的資料，之後將前面預處理好的vector分別存到d_pos跟d_pre各自的expr中，稍後可在個別的child theard中進行處理。

創造t_pos跟t_pre兩個新的thread去平行處理兩種不同的運算，進入這兩個對應的函數f_pos跟f_pre後，先記錄各自的起始時間，再取得剛剛用來記錄資料用的d_pos跟d_pre，記錄自己的tid。

之後透過bool error_dect(vector <Token> expression)各自檢查expr中的infix運算式是否有錯誤，如果有錯誤則會輸出[Child 1(2)]ERROR INPUT!來回報錯誤並終止子執行續，如正確，則f_pos用token_infixToPostfix(data->expr)轉換成後置式，再用token_calculate_Postfix(data->expr)將後置式做計算，接著再將後置式及運算結果在此子執行續中印出，最後取得此執行續的總執行時間存回d_pos後結束運行，f_pre運行過程雷同但使用的function是用來做前置式的轉換及運算用的。

Main thread等待兩個child thread都結束後，透過d_pos跟d_pre中存的tid及used_time印出兩個child thread的tid跟used_time，最後輸出Main thread自己所用的時間。

完成功能:

- 基本功能全部
- 無大數運算

詳細解釋:

標頭檔:

```
#include <iostream> //io
#include <time.h>  //計算時間
#include <pthread.h>//thread
#include <string> //字串處理
#include <vector> //主要儲存容器
#include <cmath> //字串轉數字用
#include <algorithm>//反轉vector用

//開檔及取tid用
#include <bits/stdc++.h>
#include <unistd.h>
#include <sys/types.h>
#include "sys/syscall.h"
```

自訂結構:

```
//用來存個別的Token
struct Token{
  string row;//原始資料
  string proporty;//資料類別
};

//用來存Thread間需要用的資料
typedef struct my_data {
  vector <Token> expr;//存多項式
  unsigned long long int result; //存結果
  long double used_time; //存所用的時間
  int tid;//存tid
} my_data;
```

常用函式:

```
//將字串轉成數字
unsigned long long int str_to_int(string _in){
  unsigned long long temp = 0;
  unsigned long long length = _in.length();
  for(int i = length; i > 0; i--){
    temp += ((unsigned long long)(_in[i-1])-48)*pow(10,length-i);
  }
  return temp;
}
//判斷是不是Operator
bool t_isOperator(Token c){
  if(c.proporty ==  "SUBTRACT_TOKEN" || c.proporty ==  "ADD_TOKEN" || c.proporty ==  "MUTIPLY_TOKEN" || c.proporty ==  "DEVIDE_TOKEN")
  else return false;
}
//判斷是不是Operand
bool t_isOperand(Token c){
    if(c.proporty ==  "NUMBER_TOKEN")return true;
  else return false;
}
//取得運算的優先權
int t_getPriority(Token C){
    if (C.proporty == "SUBTRACT_TOKEN" || C.proporty == "ADD_TOKEN")
    return 1;
    else if (C.proporty == "MUTIPLY_TOKEN" || C.proporty == "DEVIDE_TOKEN")
        return 2;
    return 0;
}
//印出表達式
void print_expr(vector <Token> expr){
  for(int i = 0 ; i < int(expr.size()); ++i){
    cout << expr[i].row << ' ';
  }
  cout << "= ";
}
```

Reader成員函式介紹:

```
void make_token()//產生Token並存到vector <Token> token中

void Error()//當讀到未知的字符時會產生"SYNTAX ERROR"

void print_token()//輸出vector <Token> token到螢幕上 (檢查用)
```

```
vector <Token> get_token() //取得vector <Token> token

int get_line(){return line;}//取得檔案行數(無實裝)
```

infix轉postfix:

```
//用Stack的概念實作

vector <Token> token_infixToPostfix(vector <Token> infix){
  vector <Token> opr;
  vector <Token> output;

  for (int i = 0; i < infix. size(); i++) {
    if (t_isOperand(infix[i])){
      output.push_back(infix[i]);
    }
    else if (infix[i].proporty == "LEFTPAREN_TOKEN"){
      opr.push_back(infix[i]);
    }
    else if (infix[i].proporty == "RIGHTPAREN_TOKEN") {
        while (opr.back().proporty != "LEFTPAREN_TOKEN") {
            output.push_back(opr.back());
            opr.pop_back();
        }
        opr.pop_back();
    }
    else{
      if (opr.size() > 0 && t_isOperator(opr.back())){
        while (opr.size() > 0 && t_getPriority(infix[i]) < t_getPriority(opr.back())){
              output.push_back(opr.back());
          opr.pop_back();
              }
            }
            opr.push_back(infix[i]);
    }
  }
  while(!opr.empty()){
    output.push_back(opr.back());
    opr.pop_back();
  }
  return output;
}
```

計算Postfix:

```
//一個stack放結果，一個放Operator
long double token_calculate_Postfix(vector <Token> post_exp){
    stack <long double> stack;
    int len = post_exp.size();
    for (int i = 0; i < len; i++){
        if ( t_isOperand(post_exp[i])){
            stack.push(str_to_int(post_exp[i].row));
        }
        else{
            long double a = stack.top();
            stack.pop();
            long double b = stack.top();
            stack.pop();
            switch (post_exp[i].row[0]){
                case '+': // addition
                          stack.push(b + a);
                          break;
                case '-': // subtraction
                          stack.push(b - a);
                          break;
                case '*': // multiplication
                          stack.push(b * a);
                          break;
                case '/': // division
                          stack.push(b / a);
                          break;
            }
        }
    }
    return stack.top();
}
```

infix轉prefix:

```
//顛倒，做postfix(左右括號相反)，再顛倒
vector <Token> token_infixToPrefix(vector <Token> infix){
  vector <Token> output;
  vector <Token> opr;
  reverse(infix.begin(), infix.end());
  for (int i = 0; i < infix. size(); i++) {
    if (t_isOperand(infix[i])){
      output.push_back(infix[i]);
    }
    else if (infix[i].proporty == "RIGHTPAREN_TOKEN"){
      opr.push_back(infix[i]);
    }
    else if (infix[i].proporty == "LEFTPAREN_TOKEN") {
        while (opr.back().proporty != "RIGHTPAREN_TOKEN") {
            output.push_back(opr.back());
            opr.pop_back();
        }
        opr.pop_back();
    }
    else{
      if (opr.size() > 0 && t_isOperator(opr.back())){
        while (opr.size() > 0 && t_getPriority(infix[i]) < t_getPriority(opr.back())){
              output.push_back(opr.back());
              opr.pop_back();
        }
      }
            opr.push_back(infix[i]);

    }
  }

        while(!opr.empty()){
          output.push_back(opr.back());
          opr.pop_back();
      }
  reverse(output.begin(), output.end());
  return output;
}
```

計算prefix:

```
//從後往前算
long double token_evaluatePrefix(vector <Token> exprsn)
{
    stack<long double> Stack;
    for (int j = exprsn.size() - 1; j >= 0; j--) {
        if (t_isOperand(exprsn[j])){
            Stack.push(str_to_int(exprsn[j].row));
        }
        else {
            long double o1 = Stack.top();
            Stack.pop();
            long double o2 = Stack.top();
            Stack.pop();
            switch (exprsn[j].row[0]) {
            case '+':
                Stack.push(o1 + o2);
                break;
            case '-':
                Stack.push(o1 - o2);
                break;
            case '*':
                Stack.push(o1 * o2);
                break;
            case '/':
                Stack.push(o1 / o2);
                break;
            }
        }
    }
    return Stack.top();
}
```

infix算式錯誤偵測:

```
//似做postfix，但主要計算搭配跟括號，return true代表有問題，return false代表沒問題
bool error_dect(vector <Token> expression){
        bool result = false;
        vector <Token> output;
        vector <Token> opr;
        vector <int> st3;
        bool isTrue = false;
        Token temp;
        bool flag;
        for (int i = 0; i < expression.size(); i++) {
            temp = expression[i];
            if (i == expression.size() && st3.size() > 0){
          return true;
        }
            if (t_isOperand(temp)) {
                output.push_back(temp);
                if(!isTrue) {
                    isTrue = true;
                }
                else {
                    return true;
                }
            }
            else if (t_isOperator(temp)) {
                opr.push_back(temp);
                isTrue = false;
            }else if (temp.proporty == "LEFTPAREN_TOKEN"){
      opr.push_back(temp);
      st3.push_back(1);
      }else if (temp.proporty == "RIGHTPAREN_TOKEN") {
        if(st3.size() == 0){return true;}
        st3.pop_back();
      }else{
      }
        }
        if(st3.size() > 0){return true;}
        return result;
}
```

child thread所對應的函式:

```
//對應Postfix
void *f_pos(void *arg) {
  clock_t start = clock();
  my_data *data=(my_data *)arg;
  data->tid = gettid();

  //error detect
  if(error_dect(data->expr)){
    cout << "[Child 2] ERROR INPUT!" << endl;
    data->used_time = difftime(clock() , start);
    pthread_exit(NULL);
  }

  data->expr = token_infixToPostfix(data->expr);
  data->result = roundl(token_calculate_Postfix(data->expr));

  cout << "[Child 2] ";
  print_expr(data->expr);
  cout << data->result << endl;


  data->used_time = difftime(clock() , start);
  pthread_exit(NULL);
}

//對應Prefix
void *f_pre(void *arg) {
  clock_t start = clock();
  my_data *data=(my_data *)arg;
  data->tid = gettid();

  //error detect
  if(error_dect(data->expr)){
    cout << "[Child 1] ERROR INPUT!" << endl;
    data->used_time = difftime(clock() , start);
    pthread_exit(NULL);
  }
  data->expr = token_infixToPrefix(data->expr);
  data->result = roundl(token_evaluatePrefix(data->expr));


  cout << "[Child 1] ";
  print_expr(data->expr);
  cout << data->result << endl;
  data->used_time = difftime(clock() , start);
  pthread_exit(NULL);
}
```

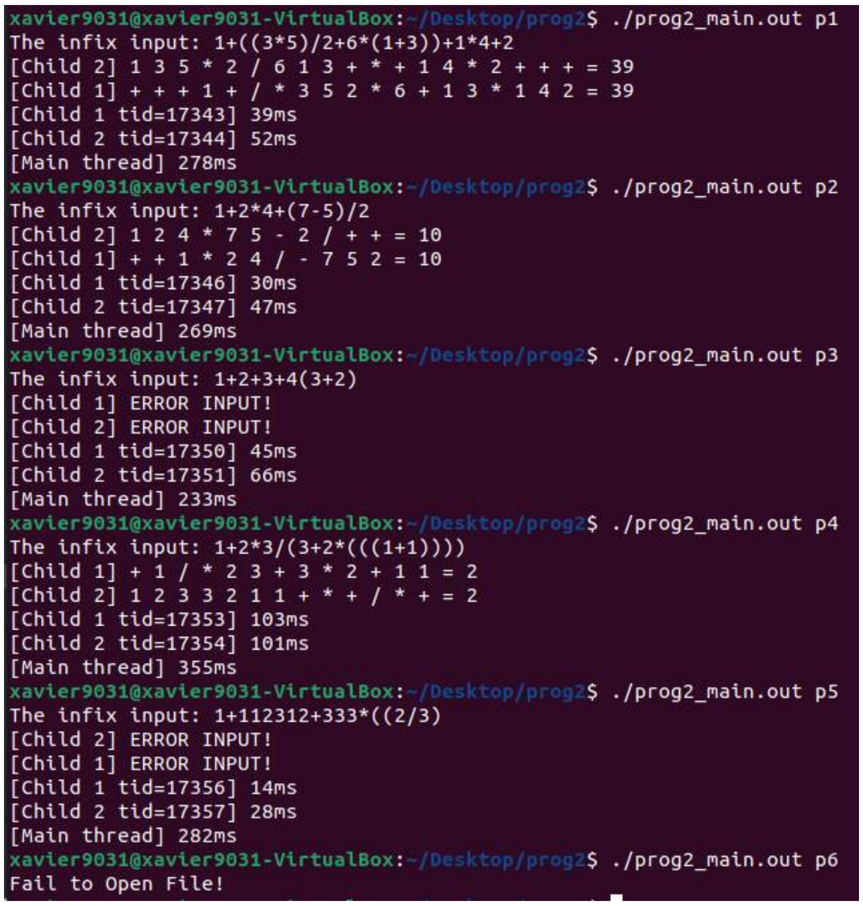
Driven Code:

```
//透過argv傳檔名並開啟
int main(int argc, char* argv[]) {
  clock_t start;
  start = clock();

  string str ;
  ifstream inFile;
  inFile.open(argv[1]);
  if(getline(inFile,str)){}
  else{
    cout << "Fail to Open File!" << endl;//開檔失敗
    return 0;
  };
  cout << "The infix input: " << str << endl;//輸出檔案內容


  Reader a(str);
  a.make_token();//做Token
  vector <Token> str_token = a.get_token();//拿Token
  pthread_t t_pos,t_pre;
  my_data d_pos, d_pre;
  d_pos.expr = str_token;
  d_pre.expr = str_token;
  vector<Token>().swap(str_token);
  //產生child thread
  pthread_create(&t_pre, NULL, f_pre, (void*) &d_pre);
  pthread_create(&t_pos, NULL, f_pos, (void*) &d_pos);
  //等child thread結束
  pthread_join(t_pre, NULL);
  pthread_join(t_pos, NULL);
  //印出執行時間跟tid
  cout << "[Child 1 tid=" << d_pre.tid << "] " << d_pre.used_time << "ms" << endl;
  cout << "[Child 2 tid=" << d_pos.tid << "] " << d_pos.used_time << "ms" << endl;
  cout << "[Main thread] " << difftime(clock() , start) << "ms" << endl;
  return 0;
}
```

More Example:



```
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ ./prog2_main.out p1
The infix input: 1+((3*5)/2+6*(1+3))+1*4+2
[Child 2] 1 3 5 * 2 / 6 1 3 + * + 1 4 * 2 + + + = 39
[Child 1] + + + 1 + / * 3 5 2 * 6 + 1 3 * 1 4 2 = 39
[Child 1 tid=17343] 39ms
[Child 2 tid=17344] 52ms
[Main thread] 278ms
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ ./prog2_main.out p2
The infix input: 1+2*4+(7-5)/2
[Child 2] 1 2 4 * 7 5 - 2 / + + = 10
[Child 1] + + 1 * 2 4 / - 7 5 2 = 10
[Child 1 tid=17346] 30ms
[Child 2 tid=17347] 47ms
[Main thread] 269ms
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ ./prog2_main.out p3
The infix input: 1+2+3+4(3+2)
[Child 1] ERROR INPUT!
[Child 2] ERROR INPUT!
[Child 1 tid=17350] 45ms
[Child 2 tid=17351] 66ms
[Main thread] 233ms
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ ./prog2_main.out p4
The infix input: 1+2*3/(3+2*(((1+1))))
[Child 1] + 1 / * 2 3 + 3 * 2 + 1 1 = 2
[Child 2] 1 2 3 3 2 1 1 + * + / * + = 2
[Child 1 tid=17353] 103ms
[Child 2 tid=17354] 101ms
[Main thread] 355ms
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ ./prog2_main.out p5
The infix input: 1+112312+333*((2/3)
[Child 2] ERROR INPUT!
[Child 1] ERROR INPUT!
[Child 1 tid=17356] 14ms
[Child 2 tid=17357] 28ms
[Main thread] 282ms
xavier9031@xavier9031-VirtualBox:~/Desktop/prog2$ ./prog2_main.out p6
Fail to Open File!
```