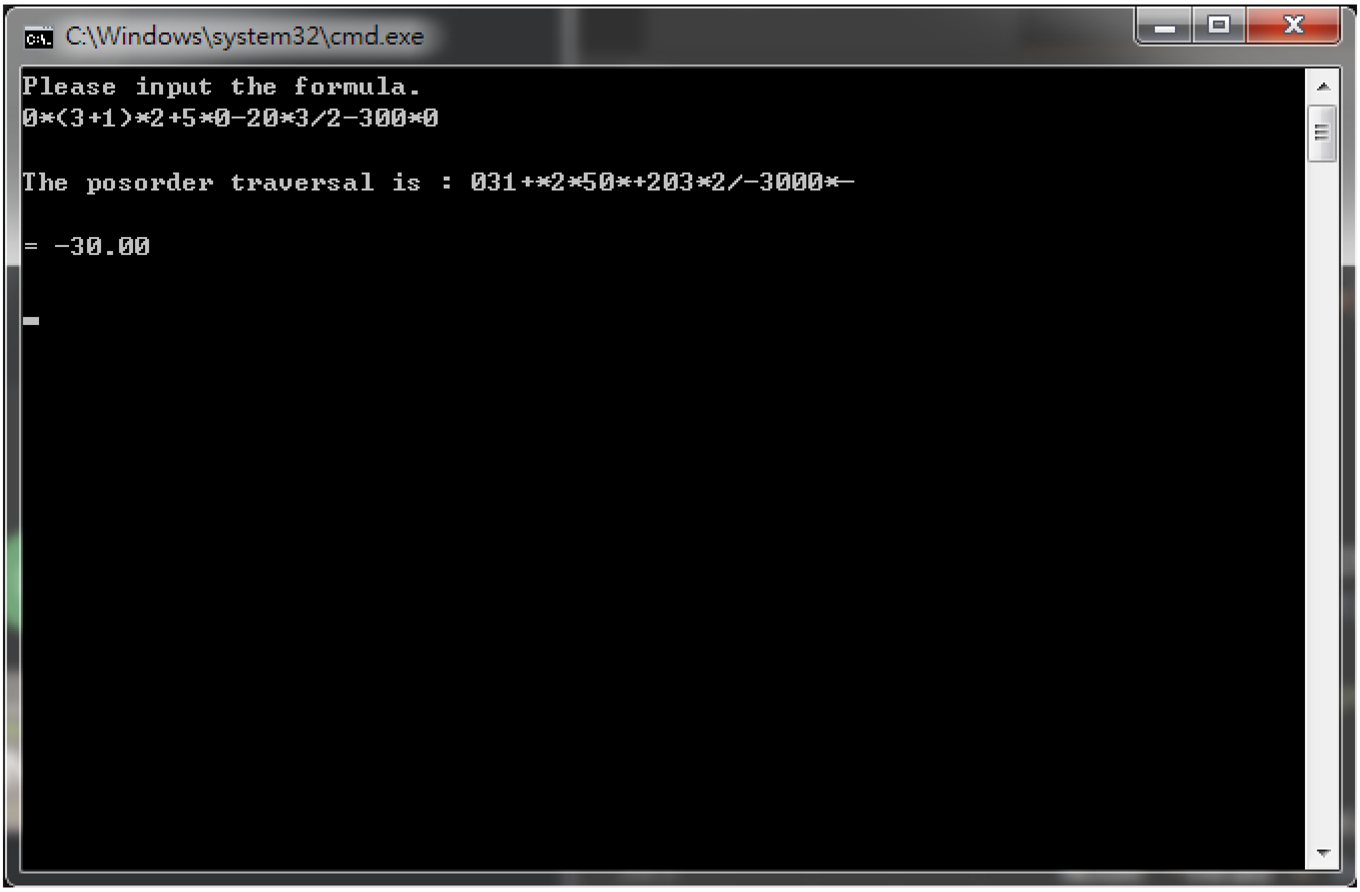
Data_Structure Hw06 Readme

struct Node

```
struct Node{
        char op;
        double value;
        Node *left, *right;
}
```

輸入介面



```
Please input the formula.
0*(3+1)*2+5*0-20*3/2-300*0

The posorder traversal is : 031+*2*50*+203*2/-3000*-

= -30.00


_
```

- 輸出 Please input the formula.
- 輸入運算式
- 輸出後序運算式
- 輸出答案
- 繼續輸入下一個運算式

Function

is_operator

```
bool is_operator(char c){
        if(c == '+' || c == '-' || c == '*' || c == '/') return true;
        return false;
}
```

- 說明: 判斷字元 c 是否為 +, -, *, /

is_number

```
bool is_number(char c){
        if(c >= '0' && c <= '9') return true;
        return false;
}
```

- 說明: 判斷字元 c 是否為 0, 1, ..., 9

oper_priority

```
bool oper_priority(char a, char b){
        if(!b) return true;

        if(a == '+' || a == '-') return true;
        else if(a == '*' || a == '/'){
                if(b == '*' || b == '/') return true;
                else return false;
        }

        return false;
}
```

- 說明: 判斷運算子 a 的優先度是否比運算子 b 高，如果比較高的話就是 true
- 注意: 因為是做二元樹，優先度比較低的運算子會在比較高，也就是說這個運算子要比較早先拆

compute_node

```
double compute_node(double a, double b, char op){
        switch(op){
                case '+':
                        return a + b;
                case '-':
                        return a - b;
                case '*':
                        return a * b;
                case '/':
                        return a / b;
        }

        return 0;
}
```

- 說明: 計算每個結點的值

node_init

```
Node *node_init(){
        Node *temp = new Node;
        temp -> op = 0;
        temp -> value = 0;
        temp -> left = NULL;
        temp -> right = NULL;
        return temp;
}
```

- 說明: node初始化

build_leaf

```
Node *build_leaf(string s, bool is_signed){
        //transfer string to int
        int64_t num = 0;
        for(int64_t i = is_signed & 1; i < s.length(); i++){
                num *= 10;
                num += s[i] - '0';
        }

        Node *temp = node_init();
        if(is_signed) temp -> value = -num;
        else temp -> value = num;

        return temp;
}
```

- 說明: 建一個葉結點

- 注意: 如果是負數的話，數字會從 index = 1 開始跑

build_node

```
build_node(string s){
  Node *temp = new Node;

  int sub_i = -1;        //record the last -'s index
  int sub_cnt = 0;       //record how many -

  //jump over index 0
  //judge whether separate strirng into two substring or not
  bool check = true;
  for(int64_t i = 1; i < s.length(); i++){
          if(is_operator(s[i])){
                  check = false;
                  if(s[i] == '-'){
                          sub_cnt += 1;
                          sub_i = i;
                  }
          }
  }
  //if noly one and the index of - is 1 ex. (-10)
  if(sub_cnt == 1 && sub_i == 1) check = true;

  //if the string can't be separate
  if(check){
          if(s[0] == '-') return build_leaf(s, true);
          else if(is_number(s[0])) return build_leaf(s, false);
          else if(s[0] == '(' && s[s.length() - 1] == ')'){
                  s.assign(s, 1, s.length() - 2);

                  if(s[0] == '-') temp = build_leaf(s, true);
                  else if(is_number(s[1])) temp = build_leaf(s, false);

                  return temp;
          }
  }

  char it = 0;    //record the separate operator
  int index = 0; //record the separate operator's index
  string sta;            //record the parentheses completeness
  for(int64_t i = 0; i < s.length(); i++){
          if(s[i] == '(') sta.push_back('(');
          else if(s[i] == ')'){
                  if(!sta.empty() && sta[sta.length() - 1] == '(') sta.pop_back();
                  else sta.push_back(')');
          }
          else if(sta.empty() && is_operator(s[i]) && oper_priority(s[i], it)){
                  it = s[i];
                  index = i;
          }
  }

  //if there are no operator can be separate ex.(1+2)
  if(!index){
          s.assign(s, 1, s.length() - 2);
          return build_node(s);
```

```
}

//build node
if(is_operator(it)){
        temp->op = s[index];

        string a, b;
        a.assign(s, 0, index);
        b.assign(s, index + 1, s.length());

        //after separating, the substring might have parenteses
        if(a[0] == '(' && a[a.length() - 1] == ')')
                a.assign(a, 1, a.length() - 2);
        if(b[0] == '(' && b[b.length() - 1] == ')')
                b.assign(b, 1, b.length() - 2);

        temp -> left = build_node(a);
        temp -> right = build_node(b);
        temp -> value = compute_node(temp -> left -> value, temp -> right -> value, ter
}

return temp;
```

- 說明: 建立二元樹
- 做法: 依照運算元分成左右兩個子字串，再由子字串建一顆二元樹，依此類推
- 注意
  - 因為要處理負數跟括號的問題，所以要紀錄這個字串裡面有幾的負號跟最後一個負號的位置在哪裡
  - 如果只有一個負號且在 index = 1 的地方，判斷變為 true
  - 可以分割的運算子要建立在括號是完整的情況下
  - 如果沒有運算子可以分割，代表這個字串是由一個完整且正確的括號包起來，所以要先把括號拆開在遞迴下去
  - 拆開的左右子字串有可能會有一組括號包起來，要先拆開

delete_node

```
Node *delete_node(Node *root){
        if(root -> left) root -> left = delete_node(root -> left);
        if(root -> right) root -> right = delete_node(root -> right);
        delete root;
        return NULL;
}
```

- 說明: 刪除二元樹

posorder_traversal

```
void posorder_traversal(Node *root){
        if(root){
                posorder_traversal(root -> left);
                posorder_traversal(root -> right);
                if(!root -> op){
                        if(root -> value < 0) printf("(%.0lf)", root -> value);
                        else printf("%.0lf", root -> value);
                }
                else printf("%c", root -> op);
        }
}
```

- 說明: 後序輸出運算式

check_formula

```
ol check_formula(string s){
        char pre = 0;
        string sta;

        for(int64_t i = 0; i < s.length(); i++){
                if(s[i] == '(') sta.push_back('(');
                if(s[i] == ')'){
                        if(sta[sta.length() - 1] == '(') sta.pop_back();
                        else sta.push_back(')');
                }

                if(pre == '(' && s[i] == ')'){
                        printf("Left parenthesis followed by a right parenthesis\n");
                        return false;
                }
                else if(pre == ')' && s[i] == '('){
                        printf("Right parenthesis followed by a left parenthesis\n");
                        return false;
                }
                else if(!is_operator(s[i]) && !is_number(s[i]) && s[i] != '(' && s[i] != ')
                        printf("Illegal character\n");
                        return false;
                }
                else if(pre == ')' && !is_operator(s[i]) && s[i] != ')'){
                        printf("Right parenthesis followed by an identifier\n");
                        return false;
                }
                else if(pre == '(' && is_operator(s[i]) && s[i] != '-'){
                        printf("Left parenthesis followed by an operator\n");
                        return false;
                }
                else if(is_operator(pre) && is_operator(s[i])){
                        printf("Operator followed by an operator\n");
                        return false;
                }
                else if(is_number(pre) && s[i] == '('){
                        printf("Identifier followed by a left parenthesis\n");
                        return false;
                }
                else if(is_operator(pre) && s[i] == ')'){
                        printf("Operator followed by a right parenthesis\n");
                        return false;
                }

                pre = s[i];
        }

        if(!sta.empty()){
                if(sta[sta.length() - 1] == '('){
                        printf("Unmatched left parenthesis\n");
                        return false;
                }
                else{
                        printf("Unmatched right parenthesis\n");
                        return false;
                }
        }

        if(is_operator(s[0])){
                printf("First character an operator\n");
                return false;
        }
        if(is_operator(s[s.length() - 1])){
                printf("Last character an operator\n");
                return false;
        }

        return true;
```

- 說明: 依照作業要求判斷輸入字串是否正確

main

```
int main(){
        ios::sync_with_stdio(false);
        cin.tie(0);

        HANDLE h;
        string formula;
        do{
                h = GetStdHandle(STD_INPUT_HANDLE);
                if(WaitForSingleObject(h, 0) == WAIT_OBJECT_0){
                        system("cls");
                        printf("Please input the formula.\n");
                        cin >> formula;

                        if(check_formula(formula)){
                                Node *head = build_node(formula);

                                printf("\n");
                                traversal_node(head, 2);
                                printf("\n");

                                printf("= %.2lf\n\n", head -> value);

                                delete_node(head);
                        }
                }
        }while(GetAsyncKeyState(VK_ESCAPE) == 0);

        return 0;
}
```