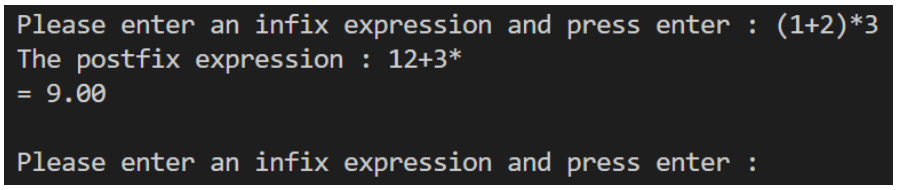
Data Structure HW6

40847013S 王瑞渝

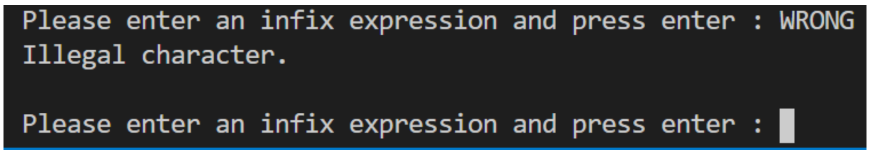
目錄

1.1 輸入



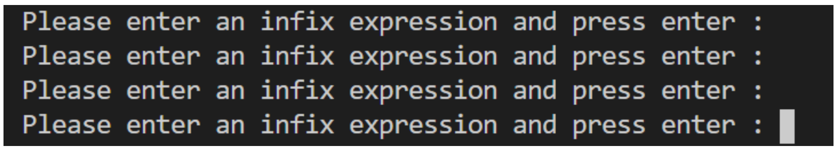
```
Please enter an infix expression and press enter : (1+2)*3
The postfix expression : 12+3*
= 9.00

Please enter an infix expression and press enter :
```

▲ 輸入合理的運算式以進行



```
Please enter an infix expression and press enter : WRONG
Illegal character.

Please enter an infix expression and press enter :
```

▲ 輸入錯誤運算式，輸出警示



```
Please enter an infix expression and press enter :
Please enter an infix expression and press enter :
Please enter an infix expression and press enter :
Please enter an infix expression and press enter :
```

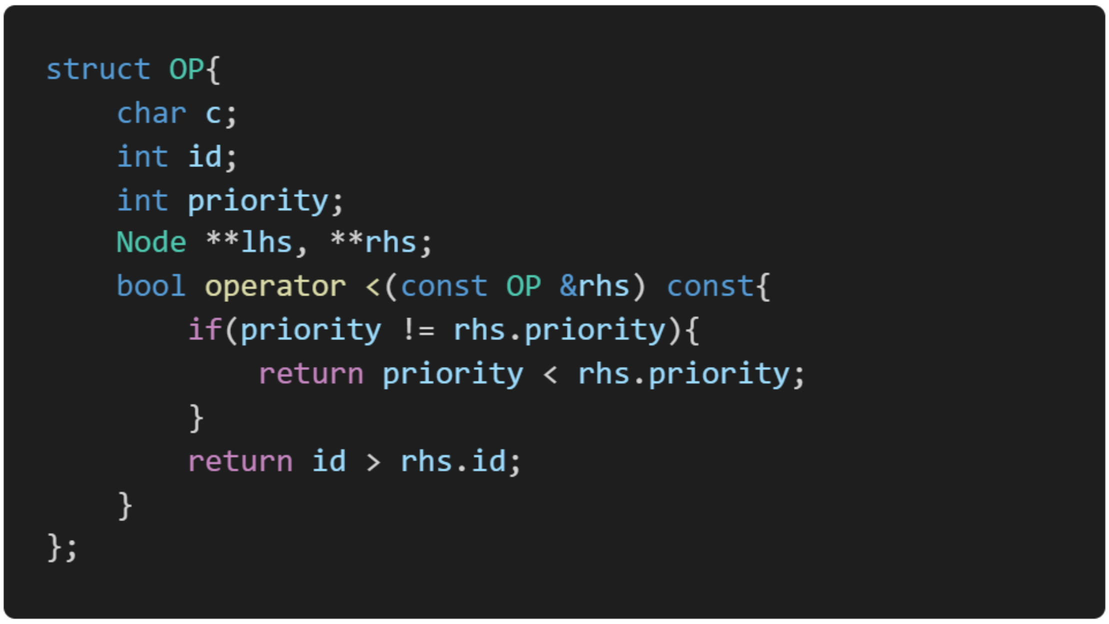
▲ 輸入為空時，不進行計算

2.1 功能分析——STRUCT NODE

```
struct Node{
    Node *parent = NULL;
    Node *lchild = NULL;
    Node *rchild = NULL;
    enum{
        TYPE_NUM,
        TYPE_OP
    } DataType;
    union{
        int num;
        char c;
    };
};
```

- parent → 父節點指標
- lchild & rchild → 子節點指標
- enum DataType → 記錄存放資料的型態
- union → 存放資料

2.2 功能分析——STRUCT OP



```
struct OP{
    char c;
    int id;
    int priority;
    Node **lhs, **rhs;
    bool operator <(const OP &rhs) const{
        if(priority != rhs.priority){
            return priority < rhs.priority;
        }
        return id > rhs.id;
    }
};
```

- c → 存放運算子
- id → 先後順序(從 0 開始)
- priority → 該運算子在整個運算式中的優先度
- **lhs & **rhs → 該運算子兩端的運算數
- operator < → priority 大的優先，相同的話 id 小的優先

2.3 功能分析——ENUM CHECKFLAG

```
enum CheckFlag{
    ILLEGAL_CHAR = 1,
    FIRST_OP = 2,
    LAST_OP = 4,
    RP_I = 8,
    OP_OP = 16,
    LP_OP = 32,
    I_LP = 64,
    RP_LP = 128,
    OP_RP = 256,
    LP_RP = 512,
    UNMATCHED_LP = 1024,
    UNMATCHED_RP = 2048,
    MORE_THAN_20 = 4096
};
```

使用 bit 運算去判斷錯誤 Flag。

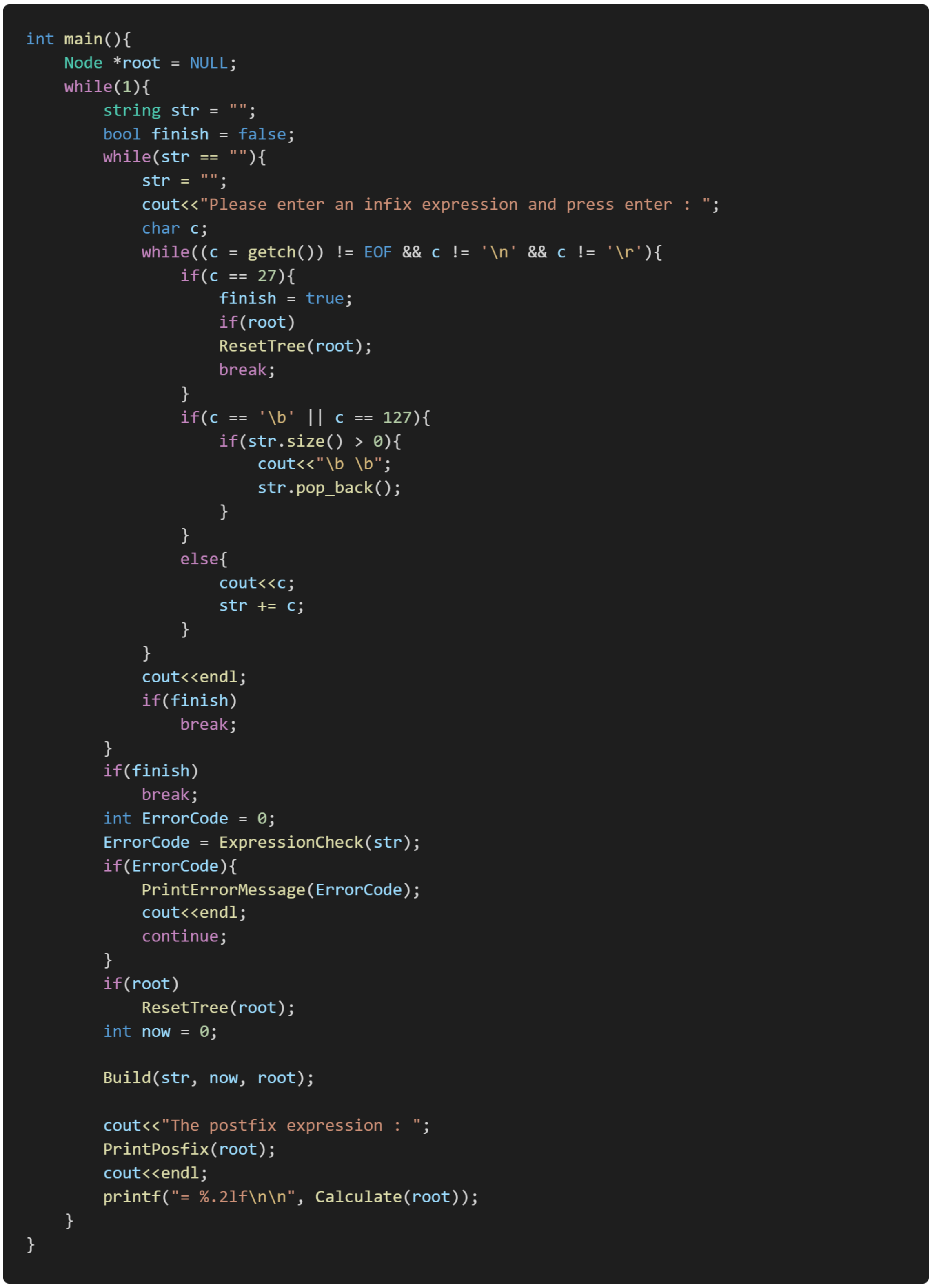
2.4 功能分析——ESC 退出&MAIN

```c
#ifdef _WIN32
#include <conio.h>
#define CLEAR "cls"
#else //In any other OS
#include <termio.h>
#define CLEAR "clear"

#ifndef STDIN_FILENO
#define STDIN_FILENO 0
#endif

int getch(void)
{
        struct termios tm, tm_old;
        int fd = STDIN_FILENO, c;
        if(tcgetattr(fd, &tm) < 0)
                return -1;
        tm_old = tm;
        cfmakeraw(&tm);
        if(tcsetattr(fd, TCSANOW, &tm) < 0)
                return -1;
        c = fgetc(stdin);
        if(tcsetattr(fd, TCSANOW, &tm_old) < 0)
                return -1;
        return c;
}
#endif
```

支援多系統，Windows 使用 conio.h 內建 getch，POSIX 則使用 termio.h 編寫。

當 getch()讀到 ESC(27)時退出程式、backspace 清除上一個字元、換行退出迴圈

之後是偵錯>重置樹>建樹>輸出 posfix>計算



```cpp
int main(){
    Node *root = NULL;
    while(1){
        string str = "";
        bool finish = false;
        while(str == ""){
            str = "";
            cout<<"Please enter an infix expression and press enter : ";
            char c;
            while((c = getch()) != EOF && c != '\n' && c != '\r'){
                if(c == 27){
                    finish = true;
                    if(root)
                    ResetTree(root);
                    break;
                }
                if(c == '\b' || c == 127){
                    if(str.size() > 0){
                        cout<<"\b \b";
                        str.pop_back();
                    }
                }
                else{
                    cout<<c;
                    str += c;
                }
            }
            cout<<endl;
            if(finish)
                break;
        }
        if(finish)
            break;
        int ErrorCode = 0;
        ErrorCode = ExpressionCheck(str);
        if(ErrorCode){
            PrintErrorMessage(ErrorCode);
            cout<<endl;
            continue;
        }
        if(root)
            ResetTree(root);
        int now = 0;

        Build(str, now, root);

        cout<<"The postfix expression : ";
        PrintPosfix(root);
        cout<<endl;
        printf("= %.2lf\n\n", Calculate(root));
    }
}
```

2.5 功能分析——偵錯

```cpp
int ExpressionCheck(string &str){
    int ErrorCode = 0;
    if(str.size() > 20){
        ErrorCode |= MORE_THAN_20;
    }
    if(strspn(str.c_str(), "1234567890()+-*/") != str.size()){
        ErrorCode |= ILLEGAL_CHAR;
    }
    if(isOP(str[0])){
        ErrorCode |= FIRST_OP;
    }
    if(isOP(str[str.size() - 1])){
        ErrorCode |= LAST_OP;
    }
    int p = 0;
    for(int i = 0; i < str.size() - 1; i++){
        if(!(ErrorCode & RP_I) && str[i] == ')' && isdigit(str[i + 1]))
            ErrorCode |= RP_I;

        if(!(ErrorCode & OP_OP) && isOP(str[i]) && isOP(str[i + 1]))
            ErrorCode |= OP_OP;

        if(!(ErrorCode & LP_OP) && str[i] == '(' && isOP(str[i + 1]) && str[i + 1] != '-')
            ErrorCode |= LP_OP;

        if(!(ErrorCode & I_LP) && isdigit(str[i]) && str[i + 1] == '(')
            ErrorCode |= I_LP;

        if(!(ErrorCode & RP_LP) && str[i] == ')' && str[i + 1] == '(')
            ErrorCode |= RP_LP;

        if(!(ErrorCode & OP_RP) && isOP(str[i]) && str[i + 1] == ')')
            ErrorCode |= OP_RP;

        if(!(ErrorCode & LP_RP) && str[i] == '(' && str[i + 1] == ')')
            ErrorCode |= LP_RP;

        if(str[i] == '(')
            p++;
        else if(str[i] == ')'){
            if(--p < 0){
                p = 0;
                ErrorCode |= UNMATCHED_RP;
            }
        }
    }
    if(str[str.size() - 1] == '(')
        p++;
    else if(str[str.size() - 1] == ')'){
        if(--p < 0){
            p = 0;
            ErrorCode |= UNMATCHED_RP;
        }
    }
    if(p > 0)
        ErrorCode |= UNMATCHED_LP;
    return ErrorCode;
}
```

```cpp
void PrintErrorMessage(int ErrorCode){
    if(ErrorCode & MORE_THAN_20)
        cout<<"Error - line contains more characters than allowed."<<endl;

    if(ErrorCode & ILLEGAL_CHAR)
        cout<<"Illegal character."<<endl;

    if(ErrorCode & FIRST_OP)
        cout<<"First character an operator."<<endl;

    if(ErrorCode & LAST_OP)
        cout<<"Last character an operator."<<endl;

    if(ErrorCode & RP_I)
        cout<<"Right parenthesis followed by an identifier."<<endl;

    if(ErrorCode & OP_OP)
        cout<<"Operator followed by an operator."<<endl;

    if(ErrorCode & LP_OP)
        cout<<"Left parenthesis followed by an operator."<<endl;

    if(ErrorCode & I_LP)
        cout<<"Identifier followed by a left parenthesis."<<endl;

    if(ErrorCode & RP_LP)
        cout<<"Right parenthesis followed by a left parenthesis."<<endl;

    if(ErrorCode & OP_RP)
        cout<<"Operator followed by a right parenthesis."<<endl;

    if(ErrorCode & LP_RP)
        cout<<"Left parenthesis followed by a right parenthesis."<<endl;

    if(ErrorCode & UNMATCHED_LP)
        cout<<"Unmatched left parenthesis."<<endl;

    if(ErrorCode & UNMATCHED_RP)
        cout<<"Unmatched right parenthesis."<<endl;

    return;
}
```

若檢測出錯誤則使用 Flag 將 ErrorCode 的對應 bit 調成 1，後面再一一檢查 bit，若為 1 則印出對應錯誤訊息。

2.6 功能分析——建樹

```
void Build(string &str, int &now, Node *&root){
    priority_queue <OP> pq;
    vector <Node*> operand(20);
    int priority = 0;
    OP tmp;
    while(now < str.size()){
        if(str[now] == '(' && str[now + 1] != '-'){
            priority += 2;
            now++;
        }
        else if(str[now] == ')'){
            priority -= 2;
            now++;
        }
        else if(isOP(str[now])){
            tmp.c = str[now];
            tmp.id = pq.size();
            tmp.priority = priority;
            if(str[now] == '*' || str[now] == '/')
                tmp.priority++;
            tmp.lhs = &operand[tmp.id];
            tmp.rhs = &operand[tmp.id + 1];
            pq.push(tmp);
            now++;
        }
        else{
            MakeLeaf(str, now, operand[pq.size()]);
        }
    }
    if(pq.empty()){
        root = operand.front();
    }
    else{
        while(!pq.empty()){
            OP op = pq.top();
            pq.pop();
            root = MergeBranch(op);
        }
    }
}
```

```cpp
void MakeLeaf(string &str, int &now, Node *&whence){
    whence = new Node;
    whence->DataType = Node::TYPE_NUM;
    if(str[now] == '('){
        now += 2;
        whence->num = (-1) * getNumFromStr(str, now);
        now++;
    }
    else{
        whence->num = getNumFromStr(str, now);
    }
    return;
}
```

```cpp
int getNumFromStr(string &str, int &now){
    int ret = 0;
    while(now < str.size() && isdigit(str[now])){
        ret = ret * 10 + (str[now] - '0');
        now++;
    }
    return ret;
}
```

```
Node* MergeBranch(OP op){
    Node *root = new Node;
    root->DataType = Node::TYPE_OP;
    root->c = op.c;
    Node *lhs = *op.lhs, *rhs = *op.rhs;

    while(lhs->parent)
        lhs = lhs->parent;
    while(rhs->parent)
        rhs = rhs->parent;
    root->lchild = lhs;
    lhs->parent = root;
    root->rchild = rhs;
    rhs->parent = root;
    return root;
}
```

首先先把對應的運算數製作成葉節點並保存進 operand，同時依照順序訂定運算子的 id 和 priority 製作成 OP 丟進 priority_queue (省 sort)。

如果沒有運算子(pq.empty())則將 root 設為唯一存在的運算數。

有運算子則按照大小順序依次連接各子樹。

2.7 功能分析——砍樹

```
void ResetTree(Node *root){
    if(root->lchild)
        ResetTree(root->lchild);
    if(root->rchild)
        ResetTree(root->rchild);
    free(root);
    root = NULL;
    return;
}
```

遞迴 free 節點。

2.8 功能分析──後序輸出

```cpp
void PrintPosfix(Node *root){
    if(root->lchild)
        PrintPosfix(root->lchild);
    if(root->rchild)
        PrintPosfix(root->rchild);
    if(root->DataType == Node::TYPE_NUM){
        if(root->num < 0){
            cout<<'('<<root->num<<')';
        }
        else{
            cout<<root->num;
        }
    }
    else{
        cout<<root->c;
    }
    // cout<<" ";
    return;
}
```

後序遞迴，並在 return 前根據 DataType 印出資料。

2.9 功能分析——計算

```
double Calculate(Node *root){
    if(root->DataType == Node::TYPE_NUM){
        return (double)root->num;
    }
    if(root->c == '+'){
        return Calculate(root->lchild) + Calculate(root->rchild);
    }
    else if(root->c == '-'){
        return Calculate(root->lchild) - Calculate(root->rchild);
    }
    else if(root->c == '*'){
        return Calculate(root->lchild) * Calculate(root->rchild);
    }
    else if(root->c == '/'){
        return Calculate(root->lchild) / Calculate(root->rchild);
    }
    return 0;
}
```

如果該節點的 DataType 為 TYPE_NUM 的話(=葉節點)，回傳數字。

如果該節點為運算子的話，根據預算子計算下面子樹的遞迴回傳值。