執行環境：Windows

程式語言：C

程式內容說明：

在evaluate_prefix及infix_to_prefix定義以下結構：

```
typedef enum
{rightparenthesis,leftparenthesis,times,divide,plus,minus,mod,operand}precedence;
typedef struct
{
int data;          //在infix_to_prefix中data的形態為  precedence
}element;
element stack[Max_Stack_Size];
int top = -1;
void push(element item)          //在stack的最上面加入新的element
{
if(top + 1 == Max_Stack_Size) printf("Stack is full.");
else stack[++top] = item;
return;
}
element pop()                    //從stack的最上面移除一個element  並回傳
{
if(top > -1) return stack[top--];
else printf("Stack is empty");
}
```

在evaluate_prefix中：

```
int eval(char *expr,int length) //透過此函數計算結果並回傳 expr是一個prefix expression，length是最後一個字元的index
{
    precedence token;
    element temp;
    int op1,op2,digit;

    while(length != -1) //每讀過一個字元length減1，讀完整個字串length=-1時跳出（由右往左讀）
    {
                token = get_token(expr,&length); //透過 get_token讀取目前index字元的類別，參考課本範例

                if(token == space) continue; //由於運算元和運算子由空白隔開，讀到空白直接跳到下一次的讀取
                if(token == operand) //讀到運算元時，繼續讀取直到讀到空白
                {
                         digit = 1;
                         temp.data = expr[length+1]-48; //由於右往所讀，先讀到的是個位數，接下來讀到的為十位、百位
                         while(get_token(expr,&length) != space) temp.data = temp.data + (expr[length+1]-48)*(digit*=10);
                         push(temp); //讀到空白後，將累計的數值結果push到stack
                }
                else     //不是數字就移除stack最上面的兩個element存到op1,op2中
                {
                    op1 = pop().data;
                    op2 = pop().data;
                        switch(token)     //檢查token是哪一個二元運算子，計算結果並push到stack中
                        {
                                    case times:
                                         temp.data = op1 * op2;
                                         break;
                                    case divide:
                                         temp.data = op1 / op2;
                                         break;
                                    case plus:
                                         temp.data = op1 + op2;
                                         break;
                                    case minus:
                                         temp.data = op1 - op2;
                                         break;
                                    case mod:
                                         temp.data = op1 % op2;
                                         break;
                        }
                        push(temp);
                  }
    }
    return pop().data; //由於每次運算後結果都push進stack，所以最終結果也在stack中，直接pop出答案並回傳
}
```

在 infix_to_prefix 中：

擴充 precedence 的定義如下

```
typedef enum {
        rightparenthesis,leftparenthesis,logicalnot,times,divide,mod,plus,minus,
        lshift,rshift,less,greater,lessorequal,greaterorequal,equal,notequal,
        bitwiseand,bitwiseor,logicaland,logicalor,assignment,
        operand,space
}precedence;
int icp[] = {17,17,15,13,13,13,12,12,11,11,10,10,10,10,9,9,8,6,5,4,2};
int isp[] = {0 ,17,15,13,13,13,12,12,11,11,10,10,10,10,9,9,8,6,5,4,2};
```

//icp:要放進 stack 中的 token 優先度，isp:已在 stack 中的 token 優先度

```
void infix_to_prefix(char *expr,char *prefix_expr,int length) //prefix_expr儲存轉換後的結果 expr為infix expression
{
    int i,count = 0; //count:目前prefix_expr的index從0開始
    char ctemp;
    precedence token;
    element temp;

    while(length != -1) //每讀過1個字元length減1，讀完整個expr就跳出(由右往左讀)
    {
                token = get_token(expr,&length); //透過get_token讀取目前的字元類別
                if(token == space) continue;     //token為空白就跳到下一次讀取
                if(token == operand) prefix_expr[count++] = expr[length+1]; //token是數字或未知數就輸出到prefix_expr中
                else //token如果是運算子，stack為空(top=-1)或讀到token的優先度>=stack最上面token的優先度，push到stack
                {
                    temp.data = token;  //token存到暫存的element中
                    if(token == leftparenthesis) //如果token為'('，代表')'已經在stack中
                    {                            //透過put_token輸出stack中的token直到碰到')'
                            while(stack[top].data != rightparenthesis) put_token(prefix_expr,pop(),&count);
                            pop();               // 移除stack中的')'
                    }
                    else if(top == -1 || icp[token] >= isp[stack[top].data]) push(temp);
                    else  //讀到token的優先度<stack最上面的token
                    {     //透過put_token輸出stack中的token直到token的優先度>=stack[top]才push
                        while(top != -1 && icp[token] < isp[stack[top].data]) put_token(prefix_expr,pop(),&count);
                        push(temp);
                    }
                }
    }

    while(top != -1) put_token(prefix_expr,pop(),&count);  //字串讀完將剩下的運算子輸出
    for(i = 0 ; i < (count-(count%2))/2 ; i++) //因為結果是由後往前放(右往左讀)，所以最後反轉prefix_expr
    {
          ctemp = prefix_expr[i];
          prefix_expr[i] = prefix_expr[count-1-i];
          prefix_expr[count-1-i] = ctemp;
    }
}
int main()
```