```
#include "expr_int.h"
#include "expr_float.h"
#include <conio.h>
#include "stack.h"


using namespace std;

int main(){

    cout << "Expresion Calculator 2000" << endl
         << "choose precision([int]->0, [float]->1): \n>> ";
    int c;
    c=getch();

    cout << endl << endl;

    if(c-48){
        expr_float();
    }
    else{
        expr_int();
    }

    return 0;
}

```

```
#ifndef OPERAND_H
#define OPERAND_H

class operand{
public:
    int p; char o;
    operand(){}
    operand(const int pre, const char opd):
        p(pre), o(opd){}
    operand& operator()(const int pre, const char opd){
        p=pre; o=opd;
        return *this;
    }
    bool operator<(const operand& opd){
        return p<opd.p;
    }
    bool operator>(const operand& opd){
        return p>opd.p;
    }
    bool operator<=(const operand& opd){
        return p<=opd.p;
    }
    bool operator>=(const operand& opd){
        return p>=opd.p;
    }
    bool operator&&(const operand& opd){
        return p && opd.p;
    }
    bool operator&&(const int& opd){
        return p && opd;
    }
    bool operator||(const int& opd){
        return p || opd;
    }
    bool operator||(const operand& opd){
        return p||opd.p;
    }
    bool operator!(){
        return !p;
    }
    bool operator==(const operand& opd){
        return p==opd.p;
    }
    bool operator!=(const operand& opd){
        return p!=opd.p;
    }
    bool operator==(const int& m){
        return p==m;
    }
    bool operator!=(const int& m){
        return p!=m;
    }
    bool operator>(const int& m){
        return p>m;
    }
    bool operator<(const int& m){
        return p<m;
    }
    bool operator<=(const int& m){
        return p<=m;
    }
    bool operator>=(const int& m){
        return p>=m;
    }
    operand& operator=(const operand& opd){
        p=opd.p;
        o=opd.o;
        return *this;
    }

};
#endif
```

```cpp
#ifndef STACK_H
#define STACK_H

template <class t> class stack;

template <class u>
class node{
public:
    node():next(nullptr){}
    node(u m_var):next(nullptr, var(m_var)){}
private:
    u var;
    node* next;
    node& operator=(const u& n_var){
        return this->var = n_var;
    }
    friend class stack<u>;
   };

template <class t>
class stack{
private:
    int m_TOP;

    node<t>* m_current;
    node<t>* m_start;

public:

    // Overloaded Constructors Methods
    stack(int size = 0){
        m_TOP=-1;
        m_current=m_start=nullptr;
        for(int i(0); i<size; i++)
            push_back();
    }
    stack(int size, const t* array){
        m_TOP=-1;
        m_current=m_start=nullptr;
        for(int i(0); i<size; i++)
            push_back(t[i]);
    }
    stack(int size, const t var){
        m_TOP=-1;
        m_current=m_start=nullptr;
        for(int i(0); i<size; i++)
            push_back(t);
    }
    stack(const stack& obj){
        m_TOP=-1;
        m_current=m_start=nullptr;

        for(int i(0); i<=obj.size(); i++){
            push_back(obj.get(i));
        }
    }
    stack(const stack&& obj){
        m_TOP=-1;
        m_current=m_start=nullptr;
        m_start = obj[0];
        m_current = obj[obj.size()];
        obj.m_TOP = -1;
        obj.m_current = obj.m_start= nullptr;
        obj.clear();
    }

    // A Destructor Method
    ~stack(){
        clear();
    }

    // Normal Methods:
    void push_back(const t& var){
        m_TOP++;
        if(m_TOP==0){
            m_current = new node<t>;
            m_current->next = nullptr;
            m_current->var = var;
            m_start = m_current;
        } else{
            m_current->next = new node<t>;
            m_current = m_current->next;
            m_current->next = nullptr;
            m_current->var = var;
        }

    }
    void push_back(){
        m_TOP++;
        if(m_TOP==0){
            m_current = new node<t>;
            m_current->next = nullptr;
            m_start = m_current;
        } else{
            m_current->next = new node<t>;
            m_current = m_current->next;
            m_current->next = nullptr;
        }

    }
    t pop_out(){
        t var;
        if(m_TOP > -1){
            if(m_TOP==0){
                var = m_current->var;
                delete m_current;
                m_current=m_start=nullptr;
            }else if(m_TOP==1){
                var = m_start->next->var;
                delete m_start->next;
                m_start->next=nullptr;
                m_current=m_start;
            } else{
                var = m_current->var;

                node<t>* m_del = m_start;
                for(int i(0); i<(m_TOP-1); i++)
                      m_del = m_del->next;

                m_current = m_del;
                delete m_del->next;
                m_del->next=nullptr;
            }

            m_TOP--;

        }
        return var;
    }

    t get(unsigned int index){
        t var;
        node<t>* m_get = m_start;
        if(index<=m_TOP)
            for(int i(0); i<index; i++)
                m_get = m_get->next;
        var = m_get->var;
        return var;
    }
    t top(){
        if(m_TOP > -1){
        return m_current->var;
        }
    }
    int size(){
        return m_TOP;
    }
    void clear(){
        if(m_TOP!=-1){
            node<t>* m_del = m_start;
            for(int i(0); i<=m_TOP; i++){
                m_current = m_del->next;
                m_del->next = nullptr;
                delete m_del;
                m_del = m_current;
            }
        }
        m_TOP = -1;
        m_current=m_start=nullptr;
    }

    // Operator methods
    node<t>* operator[](int index){
        node<t>* m_get = m_start;
        if(index<=m_TOP)
            for(int i(0); i<index; i++)
                m_get = m_get->next;
        return m_get;
    }
    stack& operator=(const stack& obj){
        clear();

        for(int i(0); i<=obj.size(); i++){
            push_back(obj.get(i));
        }
        return this;
    }



};


#endif
```

```cpp
#include <iostream>
#include "operand.h"
#include "stack.h"
#include <math.h>

using namespace std;

#ifndef expr_int_h
#define expr_int_h

int expr_int(){

    cout << "Enter a Numeric Expression ( May include integers,(),*,/,%,^,-,+ ).";
    while(true){

        int MAXLEN(200);
        char* raw(new char[MAXLEN]);           // creating a char Array to
        cout << "\n[int]> ";                   // store user input

        cin.getline( raw , MAXLEN-1 , '\n' );  // taking input from user
        if(!strnlen(raw,MAXLEN)){              // Quit if no input
            return 0;
        }

        stack<int> postfix;                    // creating a int stack
        stack<operand> opd;                    // creating an operand stack

        opd.push_back(operand(-1,'('));        // Pushing an opening bracket

        bool error(0);                         // an error flag

        /* Following loop converts Expression to
         * postfix and calculates it: */
        for( int i=0, iflag(0); i<=strlen(raw) ; ++i ){

            //1. For a Literal
            if((int)(raw[i])-48 >= 0 && (int)(raw[i])-48 <= 9){
                if(iflag){
                    int a =(int)(raw[i])-48 + postfix.pop_out() * 10;
                    postfix.push_back(a);
                } else{
                    iflag=1;
                    int a =(int)(raw[i])-48;
                    postfix.push_back(a);
                }
            }

            //2. For an Operand
            else if(raw[i] == '(' || raw[i] == ')' ||
                    raw[i] == '*' || raw[i] == '/' ||
                    raw[i] == '%' || raw[i] == '-' ||
                    raw[i] == '+' || raw[i] == '^' ||
                    raw[i] == ' ' || raw[i] == '\0'){

                iflag=0;    //
                int poco;   // Operand priority flag

                // Sets operand priority flag
                switch(raw[i]){
                    case '+':case '-':poco=1;break;
                    case '*':case '/':case '%':poco=2;break;
                    case '^':poco=3;break;
                    case ')':poco=-2;break;
                    case '(':poco=-1;break;
                    default: poco=0;break;
                }

                operand dob(poco,raw[i]);   // New Operand type

                // priority of last operand in stack is smaller
                if( (dob > 0 && dob >= opd.top()) || dob == -1 ){
                    opd.push_back(dob);
                }

                // priority of last operand in stack is larger
                else if( dob > 0 && dob <  opd.top()){

                    // Gets the last operand in stack
                    operand poped(opd.top().p, opd.top().o);

                    // Pop until last operand in stack is of smaller priority
                    while(dob < poped){

                        opd.pop_out();  // Delete the last operand

                        int b = postfix.get(postfix.size());  // Gets the last two
                        int a = postfix.get(postfix.size()-1);  // Numbers form Postfix
                                                                // Stack to work upon

                        int r(1);                               // result variable

                        postfix.pop_out();     // Clear the last two
                        postfix.pop_out();     // Number in Postfix

                        // Work upon the Numbers
                        switch(poped.o){
                            case '+':r=a+b;break;
                            case '-':r=a-b;break;
                            case '*':r=a*b;break;
                            case '/':r=a/b;break;
                            case '%':r=a%b;break;
                            case '^':for(int i(0); i<b; i++)r*=a;break;
                            default: r=a+b;break;
                        }

                        // Push the result back in postfix stack
                        postfix.push_back(r);

                        // Get the next operand in stack
                        poped(opd.top().p, opd.top().o);
                    }

                    // Now push opernad in stack
                    opd.push_back(dob);
                }

                // operand is a closing bracket
                else if(dob == -2 || dob.o == '\0'){

                    // Same as above, only that it pops operands
                    // until an opening bracket is found
                    operand poped(opd.top().p, opd.top().o);
                    while(poped != -1){
                        opd.pop_out();
                        int b = postfix.get(postfix.size());
                        int a = postfix.get(postfix.size()-1);
                        int r(1);
                        postfix.pop_out();
                        postfix.pop_out();
                        switch(poped.o){
                            case '+':r=a+b;break;
                            case '-':r=a-b;break;
                            case '*':r=a*b;break;
                            case '/':r=a/b;break;
                            case '%':r=a%b;break;
                            case '^':for(int i(0); i<b; i++)r*=a;break;
                            default: r=a+b;break;
                        }
                        postfix.push_back(r);
                        poped(opd.top().p, opd.top().o);
                    }
                    opd.pop_out();
                }

            }

            //3. An Error
            else{
                error=1;
                cout << "-> Invalid String" << endl;
                break;
            }
        }

        if(!error)                              // Printing Answer of Expression
            cout << "=> Answer: "               // if No error is present
                 << postfix.top();

        postfix.clear();                        //Clearing the stacks for next run
        opd.clear();

        cout << endl;                           //Now Ready for another expression

    }
    return 0;
}

#endif
```

```cpp
#include <iostream>
#include "operand.h"
#include "stack.h"
#include <math.h>

using namespace std;

#ifndef expr_float_h
#define expr_float_h

int expr_float(){

    cout << "Enter a Numeric Expression ( May include integers,(),*,/,%,^,-,+ ).";
    while(true){

        int MAXLEN(200);
        char* raw(new char[MAXLEN]);           // creating a char Array to
        cout << "\n[float]> ";                   // store user input

        cin.getline( raw , MAXLEN-1 , '\n' );  // taking input from user
        if(!strnlen(raw,MAXLEN)){              // Quit if no input
            return 0;
        }

        stack<double> postfix;                 // creating a double stack
        stack<operand> opd;                    // creating an operand stack

        opd.push_back(operand(-1,'('));        // Pushing an opening bracket

        bool error(0);                         // an error flag

        /* Following loop converts Expression to
         * postfix and calculates it: */
        for( int i=0, iflag(0), dflag(0); i<=strlen(raw) ; ++i ){

            //1. For a Literal
            if((int)(raw[i])-48 >= 0 && (int)(raw[i])-48 <= 9 || raw[i]=='.'){
                if(iflag && !dflag){
                    if(!(raw[i]=='.')){
                        double a =(float)((int)(raw[i])-48) + postfix.top() * 10;
                        postfix.pop_out();
                        dflag=0;
                        postfix.push_back(a);
                    }
                    else if(raw[i]=='.'){
                        dflag=1;
                    }

                }
                else if(iflag && dflag){
                    if(!(raw[i]=='.')){
                        double a =(float)((int)(raw[i])-48)/(pow(10,dflag++))
                            + postfix.top();
                        postfix.pop_out();
                        postfix.push_back(a);
                    }
                    else if(raw[i]=='.'){
                        error=1;
                        cout << "-> Invalid String" << endl;
                        break;
                    }
                }
                else if(!iflag && !dflag){
                    iflag=1;
                    int a =(int)(raw[i])-48;
                    postfix.push_back(a);
                }
                else if(!iflag && dflag){
```

```
                error=1;
                cout << "-> Invalid String" << endl;
                break;
            }
        }

        //2. For an Operand
        else if(raw[i] == '(' || raw[i] == ')' ||
                raw[i] == '*' || raw[i] == '/' ||
                raw[i] == '%' || raw[i] == '-' ||
                raw[i] == '+' || raw[i] == '^' ||
                raw[i] == ' ' || raw[i] == '\0'){
            iflag=0;dflag=0;
            int poco;
            switch(raw[i]){
                case '+':case '-':poco=1;break;
                case '*':case '/':case '%':poco=2;break;
                case '^':poco=3;break;
                case ')':poco=-2;break;
                case '(':poco=-1;break;
                default: poco=0;break;
            }
            operand dob(poco,raw[i]);
            if( (dob > 0 && dob >= opd.top()) || dob == -1 ){
                opd.push_back(dob);
            }
            else if( dob > 0 && dob <  opd.top()){
                operand poped(opd.top().p, opd.top().o);
                while(dob < poped){
                    opd.pop_out();
                    double b = postfix.get(postfix.size());
                    double a = postfix.get(postfix.size()-1);
                    double r(1);
                    postfix.pop_out();
                    postfix.pop_out();
                    switch(poped.o){
                        case '+':r=a+b;break;
                        case '-':r=a-b;break;
                        case '*':r=a*b;break;
                        case '/':r=a/b;break;
                        case '%':r=fmod(a,b);break;
                        case '^':r=pow(a,b);break;
                        default: r=a+b;break;
                    }
                    postfix.push_back(r);
                    poped(opd.top().p, opd.top().o);
                }
                opd.push_back(dob);
            }
            else if(dob == -2 || dob.o == '\0'){
                operand poped(opd.top().p, opd.top().o);
                while(poped != -1){
                    opd.pop_out();
                    double b = postfix.get(postfix.size());
                    double a = postfix.get(postfix.size()-1);
                    double r(1);
                    postfix.pop_out();
                    postfix.pop_out();
                    switch(poped.o){
                        case '+':r=a+b;break;
                        case '-':r=a-b;break;
                        case '*':r=a*b;break;
                        case '/':r=a/b;break;
                        case '%':r=fmod(a,b);break;
                        case '^':r=pow(a,b);break;
                        default: r=a+b;break;
                    }
                    postfix.push_back(r);
```

```
                        poped(opd.top().p, opd.top().o);
                    }
                    opd.pop_out();
                }

            }
            //else
            else{
                error=1;
                cout << "-> Invalid String" << endl;
                break;
            }
        }
    if(!error)
    cout << "=> Answer: " << postfix.top();
    postfix.clear();
    opd.clear();
    cout << endl ;

    }
    return 0;
}

#endif
```