```
#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
```

```
69
70                  // priority of last operand in stack is smaller
71                  if( (dob > 0 && dob >= opd.top()) || dob == -1 ){
72                      opd.push_back(dob);
73                  }
74
75                  // priority of last operand in stack is larger
76                  else if( dob > 0 && dob <  opd.top()){
77
78                      // Gets the last operand in stack
79                      operand poped(opd.top().p, opd.top().o);
80
81                      // Pop until last operand in stack is of smaller priority
82                      while(dob < poped){
83
84                          opd.pop_out();  // Delete the last operand
85
86                          int b = postfix.get(postfix.size());  // Gets the last two
87                          int a = postfix.get(postfix.size()-1);  // Numbers form Postfix
88                                                                  // Stack to work upon
89
90                          int r(1);                               // result variable
91
92                          postfix.pop_out();     // Clear the last two
93                          postfix.pop_out();     // Number in Postfix
94
95                          // Work upon the Numbers
96                          switch(poped.o){
97                              case '+':r=a+b;break;
98                              case '-':r=a-b;break;
99                              case '*':r=a*b;break;
100                             case '/':r=a/b;break;
101                             case '%':r=a%b;break;
102                             case '^':for(int i(0); i<b; i++)r*=a;break;
103                             default: r=a+b;break;
104                         }
105
106                         // Push the result back in postfix stack
107                         postfix.push_back(r);
108
109                         // Get the next operand in stack
110                         poped(opd.top().p, opd.top().o);
111                     }
112
113                     // Now push opernad in stack
114                     opd.push_back(dob);
115                 }
116
117                 // operand is a closing bracket
118                 else if(dob == -2 || dob.o == '\0'){
119
120                     // Same as above, only that it pops operands
121                     // until an opening bracket is found
122                     operand poped(opd.top().p, opd.top().o);
123                     while(poped != -1){
124                         opd.pop_out();
125                         int b = postfix.get(postfix.size());
126                         int a = postfix.get(postfix.size()-1);
127                         int r(1);
128                         postfix.pop_out();
129                         postfix.pop_out();
130                         switch(poped.o){
131                             case '+':r=a+b;break;
132                             case '-':r=a-b;break;
133                             case '*':r=a*b;break;
134                             case '/':r=a/b;break;
135                             case '%':r=a%b;break;
136                             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
```