```
1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <math.h>
 4 :
 5: #define MAX STACK SIZE 100
 6:
 7: typedef struct {
8:
        int data[MAX STACK SIZE];
9 •
        int top;
10: }
      OperandStack:
11:
12: typedef struct {
13:
        char data[MAX_STACK_SIZE];
14:
        int top;
15: }
      OperatorStack;
16:
17: int precedence (char op) {
18:
        switch (op) {
19:
        case '+':
        case '-':
20:
21:
            return 1:
        case /*/:
22:
23:
        case '/':
24:
            return 2;
25:
26:
        return 0;
27: }
28:
29: void operandPush (OperandStack* stack, int value) {
30:
        if (stack->top >= MAX_STACK_SIZE - 1) {
31:
            printf("Operand Stack Overflow\n");
32:
            exit(EXIT_FAILURE);
33:
34:
        stack->data[++stack->top] = value;
35: 1
36:
37: int operandPop(OperandStack* stack) {
38:
        if (stack->top == -1) {
39:
            printf("Operand Stack Underflow\n");
40:
            exit (EXIT_FAILURE);
41:
42:
        return stack->data[stack->top--];
43: }
44:
45: void operatorPush(OperatorStack* stack, char op) {
46:
        if (stack->top >= MAX_STACK_SIZE - 1) {
47:
            printf("Operator Stack Overflow\n");
48:
            exit(EXIT_FAILURE);
49:
50:
        stack->data[++stack->top] = op;
51:
52:
53: char operatorPop(OperatorStack* stack) {
54:
        if (stack->top == -1) {
55:
            printf("Operator Stack Underflow\n");
56:
            exit(EXIT_FAILURE);
57:
58:
        return stack->data[stack->top--];
59:
60:
61: void printStacks (OperandStack operand_stack, OperatorStack operator_stack) {
62:
        printf("Operand Stack: ");
63:
        for (int i = 0; i <= operand_stack.top; i++) {</pre>
64:
            printf("%d ", operand_stack.data[i]);
65:
66:
        printf("\n");
67:
```

```
68:
           printf("Operator Stack: ");
   69:
           for (int i = 0; i <= operator stack.top; i++) {</pre>
  70:
               printf("%c ", operator_stack.data[i]);
  71:
  72:
           printf("\n\n");
  73: }
  74:
  75:
  76: int eval_infix(char* expression, int start, int end) {
  77:
           OperandStack operand stack = \{ .top = -1 \};
  78:
           OperatorStack operator stack = \{ .top = -1 \};
  79:
  80:
           int i:
  81:
           int num = 0;
  82:
           int operand1, operand2;
  83:
           char operator;
  84:
  85:
           for (i = start; i <= end; i++) {</pre>
  86:
               if (expression[i] == ' ') {
  87:
                   continue; // 공백 ë¬ 'ì\213\234
  88:
  89.
               else if(expression[i] >= '0' && expression[i] <= '9'){</pre>
  90:
                   num = 0:
  91:
                   while (i <= end && (expression[i] >= '0' && expression[i] <= '9')) {
  92:
                       num = num * 10 + (expression[i] - '0');
  93:
                       i++;
  94:
  95:
                   i--;
  96:
                   operandPush (&operand stack, num);
  97:
                   printStacks (operand_stack, operator_stack); // i\212 \( 203\) 235
ì¶\234ë ¥
  98:
  99:
               else if (expression[i] == '(') {
  100:
                   int count = 1;
  101:
                   int j = i + 1;
  102:
                   while (j <= end)</pre>
  103:
                        if (expression[j] == '(') {
  104:
  105:
  106:
                        else if (expression[j] == ')') {
  107:
                            count --;
  108:
                            if (count == 0) {
  109:
                                break:
  110:
  111:
  112:
                        j++;
  113:
  114:
                   operandPush(&operand_stack, eval_infix(expression, i + 1, j - 1));
 115:
 116:
                   printStacks (operand_stack, operator_stack); // i\212 \( 203\\ 235\)
ì¶\234ë ¥
 117:
               else if (expression[i] == '+' || expression[i] == '-' || expression[i] ==
 118:
'*' || expression[i] == '/') {
                    // i\230\204i\236¬ i\227°i\202°i\236\220i\235\230
 119.
i\232°i\204 i\210\234i\234\204ê°\200 i\212¤í\203\235i\227\220 i\236\210ë\212\224
1\227°1\202°1\236\220ë3´ë\213¤ 1\236\2211\235\204 ê²½1\232°1\227\220ë$\214 ê³\2041\202°
 120:
                   while (operator_stack.top != -1 && precedence(expression[i]) <=</pre>
precedence(operator_stack.data[operator_stack.top])) {
 121:
                       operand2 = operandPop(&operand_stack);
  122:
                       operand1 = operandPop(&operand_stack);
  123:
                       operator = operatorPop(&operator_stack);
  124:
  125:
                        int result;
  126:
                        switch (operator) {
  127:
                        case '+':
  128:
                            result = operand1 + operand2;
```

```
129:
                         break:
 130:
                     case '-':
 131:
                         result = operand1 - operand2;
 132:
                         break;
 133:
                     case '*':
 134:
                         result = operand1 * operand2;
 135:
 136:
                     case '/':
 137:
                         result = operand1 / operand2;
 138:
                         break:
 139:
 140:
                     operandPush(&operand_stack, result);
 141:
                     ì¶\234ë ¥
 142:
 143:
                  operatorPush(&operator_stack, expression[i]);
 144:
                  printStacks(operand_stack, operator_stack); // i\212 mi\203\235
ì¶\234ë ¥
 145:
 146:
 147:
 148:
          // ë\202"i\235\200 i\227°i\202°i\236\220 i²\230ë|¬
          while ((operator_stack.top != -1)) {
 149:
 150:
              operand2 = operandPop(&operand_stack);
 151:
              operand1 = operandPop(&operand_stack);
 152:
              operator = operatorPop(&operator_stack);
 153:
 154:
              int result;
 155:
              switch (operator) {
 156:
              case '+':
 157:
                  result = operand1 + operand2;
 158:
                 break;
              case '-':
 159:
                  result = operand1 - operand2;
 160:
 161:
                  break;
 162:
              case '*':
 163:
                  result = operand1 * operand2;
 164:
                  break;
 165:
              case '/':
 166:
                  result = operand1 / operand2;
 167:
                  break;
 168:
 169:
              operandPush(&operand_stack, result);
 170:
              if (operator_stack.top != -1) {
 171:
              printStacks(operand_stack, operator_stack); // i\212¤í\203\235 i¶\234ë ¥
 172:
 173:
 174:
 175:
 176:
          return operand_stack.data[operand_stack.top];
 177: }
 178:
 179: int main() {
 180:
          char infix[100];
 181:
 182:
          printf("Enter the infix expression: ");
 183:
          scanf("%[^\n]s", infix);
 184:
 185:
          int length = 0;
 186:
          while (infix[length] != '\0'){
 187:
                  length ++;
 188:
 189:
          length -= 1;
 190:
 191:
          int result = eval_infix(infix, 0, length);
 192:
          printf("Result: %d\n", result);
 193:
```

```
2/2
         return 0:
195: }
196:
```