```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#define MAX_EXPR_SIZE 100
#define MAX_STACK_SIZE 100
char expr[MAX_EXPR_SIZE];
int isp[] = { 0, 19, 12, 12, 13, 13, 13, 13, 0 };
int icp[] = { 20, 19, 12, 12, 13, 13, 13, 13, 0 };

typedef enum {
    lparen, rparen, plus, minus, times, divide, mod, space, eos, operand
} precedence;
typedef struct Node* Pointer;
typedef struct Node {
    precedence data;
    Pointer next;
}Node;

Pointer top;

void rExpr(FILE* fp, char* expr);
void push(Pointer* top, precedence data);
precedence pop(Pointer* top);
precedence getToken(char* symbol, int* n);
void printToken(precedence token);
void postfix();

void main(void) {
    FILE* fp = NULL;
    rExpr(fp, expr);
    postfix();
    return;
}

void rExpr(FILE* fp, char* expr) {
    fp = fopen("expr.txt", "r");
    if (fp == NULL) {
        printf("파일 열기를 실패하였습니다.");
        exit(-1);
    }
    fgets(expr, MAX_EXPR_SIZE, fp);
    fclose(fp);
}

void push(Pointer* top, precedence data) {
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = data;
    temp->next = *top;
    *top = temp;
}

precedence pop(Pointer* top) {
    if ((*top)->next == NULL) {
        //printf("Stack Underflow₩n");
        exit(EXIT_FAILURE);
    }
```

```
57      precedence data;
58      Pointer pointer;
59
60      pointer = *top;
61      data = pointer->data;
62
63      *top = pointer->next;
64      free(pointer);
65
66      return data;
67  }
68
69  precedence getToken(char* symbol, int* n) {
70      /*get the next token,
71          symbol is the character representation, which is returned,
72          the token is represented by its enumerated value, which returned in the
              function name*/
73      *symbol = expr[(*n)++];
74      switch (*symbol) {
75      case '(': return lparen;
76      case ')': return rparen;
77      case '+':return plus;
78      case '-': return minus;
79      case '/': return divide;
80      case '*': return times;
81      case '%': return mod;
82      case ' ': return space;
83      case '₩0': return eos;
84      default: return operand; /* no error checking, default is operand*/
85      }
86  }
87
88  void printToken(precedence token) {
89      switch (token) {
90      case lparen: printf("( ");  break;
91      case rparen: printf(") ");  break;
92      case plus: printf("+ ");        break;
93      case minus: printf("- ");   break;
94      case divide: printf("/ ");  break;
95      case times: printf("* ");       break;
96      case mod: printf("% ");     break;
97      case space: break;
98      case eos: break;
99      default: printf("₩n ");
100     }
101 }
102
103 void postfix() {
104     /*output the postfix of the expression. The expression string, stack, and the top
          are global*/
105     char symbol;
106     int n = 0;
107
108     precedence token;
109
110     top = (Pointer)malloc(sizeof(Node));
```

```
111     top->data = eos;
112     top->next = NULL;
113
114     for (token = getToken(&symbol, &n); token != eos; token = getToken(&symbol, &n)) ↵
          {
115         if (token == operand) {
116             printf("%c ", symbol);
117         }
118         else if (token == rparen) {
119             while (top->data != lparen) {
120                 printToken(pop(&top));
121             }
122             pop(&top);
123         }
124         else if (token == space) continue;
125         else {
126             /* remove and print symbols whose isp is greater than or equal...
127                 ...to the current token's icp */
128             while (isp[top->data] >= icp[token]) {
129                 printToken(pop(&top));
130             }
131             push(&top, token);
132         }
133     }
134     while ((token = pop(&top)) != eos)
135         printToken(token);
136     printf("₩n");
137 }
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
```

166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181