

1. Write a c program for converting infix to postfix.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #include <string.h>
5 #define MAX_SIZE 100
6 int is_operator(char c) {
7     return (c == '+' || c == '-' || c == '*' || c == '/');
8 }
9 int precedence(char c) {
10     if (c == '+' || c == '-')
11         return 1;
12     else if (c == '*' || c == '/')
13         return 2;
14     return 0;
15 }
16 void infix_to_postfix(char infix[], char postfix[]) {
17     int i = 0, j = 0;
18     char stack[MAX_SIZE];
19     int top = -1;
20     while (infix[i] != '\0') {
21         if (isdigit(infix[i]) || isalpha(infix[i])) {
22             postfix[j++] = infix[i];
23         } else if (infix[i] == '(') {
24             stack[++top] = '(';
25         } else if (infix[i] == ')') {
26             while (top != -1 && stack[top] != '(') {
27                 postfix[j++] = stack[top--];
28             }
29             if (top == -1) {
30                 fprintf(stderr, "Error: Mismatched parentheses\n");
31                 exit(EXIT_FAILURE);
32             }
33             top--; // Pop '(' from stack
34         } else if (is_operator(infix[i])) {
35             while (top != -1 && stack[top] != '(' && precedence(stack[top]) >= precedence(infix[i])) {
36                 postfix[j++] = stack[top--];
37             }
38             stack[++top] = infix[i];
39         } else {
40             fprintf(stderr, "Error: Invalid infix expression\n");
41             exit(EXIT_FAILURE);
42         }
43         i++;
44     }
45     while (top != -1) {
46         if (stack[top] == '(') {
47             fprintf(stderr, "Error: Mismatched parentheses\n");
48             exit(EXIT_FAILURE);
49         }
50         postfix[j++] = stack[top--];
51     }
52     postfix[j] = '\0';
53 }
54 int main() {
55     char infix[MAX_SIZE];
56     char postfix[MAX_SIZE];
57     printf("Enter infix expression: ");
58     fgets(infix, sizeof(infix), stdin);
59     infix[strcspn(infix, "\n")] = '\0';
60     infix_to_postfix(infix, postfix);
61     printf("Infix expression: %s\n", infix);
62     printf("Postfix expression: %s\n", postfix);
63     return 0;
64 }
65
```

Output

```
/tmp/TV800W5ngj.o
Enter infix expression: ((a+b)-c*(d/e))+f
Infix expression: ((a+b)-c*(d/e))+f
Postfix expression: ab+cde/*-f+

=== Code Execution Successful ===
```

```
main.c
33- top--; // Pop '(' from stack
34- } else if (is_operator(infix[i])) {
35-     while (top != -1 && stack[top] != '(' && precedence(stack[top]) >= precedence(infix[i])) {
36-         postfix[j++] = stack[top--];
37-     }
38-     stack[++top] = infix[i];
39- } else {
40-     fprintf(stderr, "Error: Invalid infix expression\n");
41-     exit(EXIT_FAILURE);
42- }
43- i++;
44- }
45- while (top != -1) {
46-     if (stack[top] == '(') {
47-         fprintf(stderr, "Error: Mismatched parentheses\n");
48-         exit(EXIT_FAILURE);
49-     }
50-     postfix[j++] = stack[top--];
51- }
52- postfix[j] = '\0';
53- }
54- int main() {
55-     char infix[MAX_SIZE];
56-     char postfix[MAX_SIZE];
57-     printf("Enter infix expression: ");
58-     fgets(infix, sizeof(infix), stdin);
59-     infix[strcspn(infix, "\n")] = '\0';
60-     infix_to_postfix(infix, postfix);
61-     printf("Infix expression: %s\n", infix);
62-     printf("Postfix expression: %s\n", postfix);
63-     return 0;
64- }
65-
```

Output

```
/tmp/TV800W5ngj.o
Enter infix expression: ((a+b)-c*(d/e))+f
Infix expression: ((a+b)-c*(d/e))+f
Postfix expression: ab+cde/*-f+

=== Code Execution Successful ===
```

2. Write a c program for converting postfix to infix.

```
main.c Run Output Clear
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5 #define MAX_SIZE 100
6 struct Stack {
7     char data[MAX_SIZE];
8     int top;
9 };
10 void init_stack(struct Stack *s) {
11     s->top = -1;
12 }
13 void push(struct Stack *s, char c) {
14     if (s->top == MAX_SIZE - 1) {
15         fprintf(stderr, "Error: Stack overflow\n");
16         exit(EXIT_FAILURE);
17     }
18     s->data[++(s->top)] = c;
19 }
20 char pop(struct Stack *s) {
21     if (s->top == -1) {
22         fprintf(stderr, "Error: Stack underflow\n");
23         exit(EXIT_FAILURE);
24     }
25     return s->data[(s->top)--];
26 }
27 int is_operand(char c) {
28     return isalnum(c);
29 }
30 void postfix_to_infix(char postfix[], char infix[]) {
31     struct Stack stack;
32     init_stack(&stack);
33     int i = 0;
34     while (postfix[i] != '\0') {
```

main.c	Output
<pre>29 } 30 void postfix_to_infix(char postfix[], char infix[]) { 31 struct Stack stack; 32 init_stack(&stack); 33 int i = 0; 34 while (postfix[i] != '\0') { 35 if (is_operand(postfix[i])) { 36 push(&stack, postfix[i]); 37 } else if (postfix[i] == '+' postfix[i] == '-' postfix[i] == '*' postfix[i] == '/') { 38 char op1 = pop(&stack); 39 char op2 = pop(&stack); 40 sprintf(infix + strlen(infix),("(%c%c%c)", op2, postfix[i], op1); 41 push(&stack, infix[strlen(infix) - 1]); 42 } 43 i++; 44 } 45 while (stack.top != -1) { 46 infix[strlen(infix)] = pop(&stack); 47 } 48 infix[strlen(infix)] = '\0'; 49 } 50 int main() { 51 char postfix[MAX_SIZE]; 52 char infix[MAX_SIZE]; 53 printf("Enter postfix expression: "); 54 fgets(postfix, sizeof(postfix), stdin); 55 postfix[strlen(postfix) - 1] = '\0'; 56 postfix_to_infix(postfix, infix); 57 printf("Postfix expression: %s\n", postfix); 58 printf("Infix expression: %s\n", infix); 59 return 0; 60 } 61 }</pre>	<pre>/tmp/mEafJufbB.o Enter postfix expression: ab+c* Postfix expression: ab+c* Infix expression: @((a+b)()*c)) === Code Execution Successful ===</pre>

3. Write a c program for balancing Symbols.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define MAX_SIZE 100
5 typedef struct {
6     char data[MAX_SIZE];
7     int top;
8 } Stack;
9 void init_stack(Stack *s) {
10     s->top = -1;
11 }
12 void push(Stack *s, char c) {
13     if (s->top == MAX_SIZE - 1) {
14         fprintf(stderr, "Error: Stack overflow\n");
15         exit(EXIT_FAILURE);
16     }
17     s->data[++(s->top)] = c;
18 }
19 char pop(Stack *s) {
20     if (s->top == -1) {
21         fprintf(stderr, "Error: Stack underflow\n");
22         exit(EXIT_FAILURE);
23     }
24     return s->data[(s->top)--];
25 }
26 int is_empty(Stack *s) {
27     return s->top == -1;
28 }
29 int is_matching_pair(char open, char close) {
30     if (open == '(' && close == ')') return 1;
31     if (open == '{' && close == '}') return 1;
32     if (open == '[' && close == ']') return 1;
33     return 0;
34 }
35 int are_symbols_balanced(const char *expression) {
36     Stack stack;
37     init_stack(&stack);
38     for (int i = 0; expression[i] != '\0'; i++) {
39         char current = expression[i];
40         if (current == '(' || current == '{' || current == '[') {
41             push(&stack, current);
42         } else if (current == ')' || current == '}' || current == ']') {
43             if (is_empty(&stack) || !is_matching_pair(pop(&stack), current)) {
44                 return 0; // Not balanced
45             }
46         }
47     }
48     return is_empty(&stack);
49 }
50
51 int main() {
52     char expression[MAX_SIZE];
53     printf("Enter the expression: ");
54     fgets(expression, sizeof(expression), stdin);
55     expression[strcspn(expression, "\n")] = '\0';
56     if (are_symbols_balanced(expression)) {
57         printf("The symbols are balanced.\n");
58     } else {
59         printf("The symbols are not balanced.\n");
60     }
61     return 0;
62 }
63
```

Output

```
/tmp/kM708Sb0eb.o
Enter the expression: (a+b)#
The symbols are balanced.

=== Code Execution Successful ===
```

```
main.c
30 if (open == '(' && close == ')') return 1;
31 if (open == '{' && close == '}') return 1;
32 if (open == '[' && close == ']') return 1;
33 return 0;
34 }
35 int are_symbols_balanced(const char *expression) {
36     Stack stack;
37     init_stack(&stack);
38     for (int i = 0; expression[i] != '\0'; i++) {
39         char current = expression[i];
40         if (current == '(' || current == '{' || current == '[') {
41             push(&stack, current);
42         } else if (current == ')' || current == '}' || current == ']') {
43             if (is_empty(&stack) || !is_matching_pair(pop(&stack), current)) {
44                 return 0; // Not balanced
45             }
46         }
47     }
48     return is_empty(&stack);
49 }
50
51 int main() {
52     char expression[MAX_SIZE];
53     printf("Enter the expression: ");
54     fgets(expression, sizeof(expression), stdin);
55     expression[strcspn(expression, "\n")] = '\0';
56     if (are_symbols_balanced(expression)) {
57         printf("The symbols are balanced.\n");
58     } else {
59         printf("The symbols are not balanced.\n");
60     }
61     return 0;
62 }
63
```

Output

```
/tmp/kM708Sb0eb.o
Enter the expression: (a+b)#
The symbols are balanced.

=== Code Execution Successful ===
```