# 7주차 3차시 스택의 응용

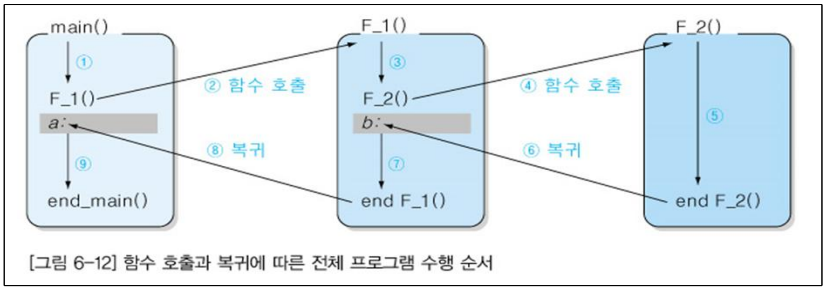
## 【학습목표】

1. 다양한 스택의 응용 방법에 대하여 설명할 수 있다.
2. 프로그램에서의 호출과 복귀에 따른 수행 순서를 설명할 수 있다.

## 학습내용1 : 시스템 스택

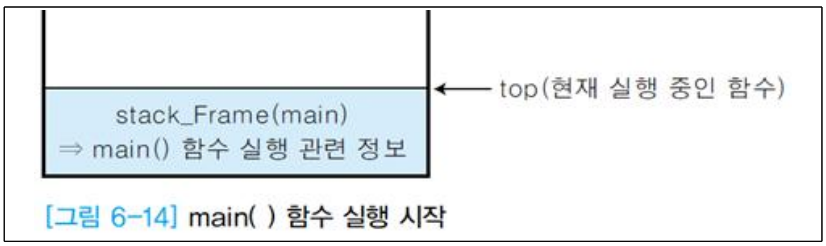
### 1. 프로그램에서의 호출과 복귀에 따른 수행 순서

- 가장 나중에 호출된 함수가 가장 먼저 실행을 완료하고 복귀하는 후입선출 구조이므로, 함수의 호출과 복귀 순서는 스택의 LIFO 구조를 응용하여 관리
- 함수 호출이 발생하면 호출된 함수 수행에 필요한 지역변수, 매개 변수 및 수행 후 복귀할 주소 등의 정보를 스택 프레임(stack frame)에 저장하여 시스템 스택에 삽입
- 시스템 스택의 top은 현재 실행중인 함수가 호출되었을 때 이에 대한 전환 정보가 들어있는 스택 프레임이 됨
- 함수의 실행이 끝나면 시스템 스택의 top 원소(스택 프레임)를 삭제(pop)하면서 스택 프레임이 저장되어 있던 복귀 주소를 확인하고 복귀
- 함수 호출과 복귀에 따라 이 과정을 반복하여 전체 프로그램 수행이 종료되면 시스템 스택은 공백 스택이 됨



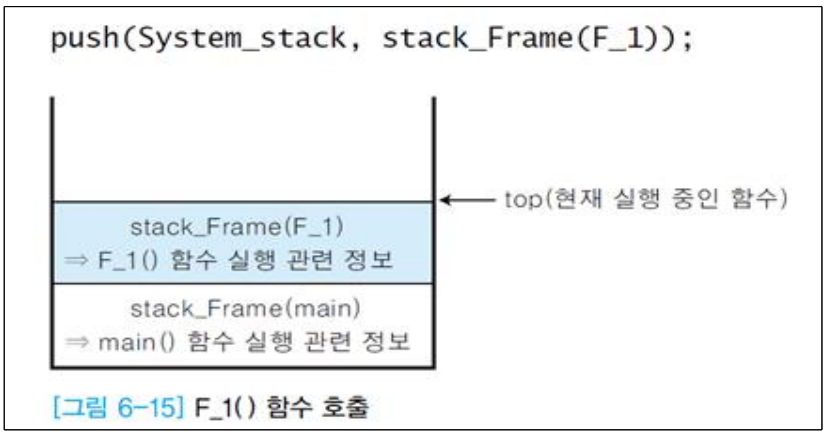
[그림 6-12] 함수 호출과 복귀에 따른 전체 프로그램 수행 순서

① 프로그램 실행을 시작하면 main()함수가 호출되어 실행되면서 mail() 함수 시작과 관련된 정보를 스택 프레임에 저장하여 시스템 스택에 삽입한다



[그림 6-14] main( ) 함수 실행 시작

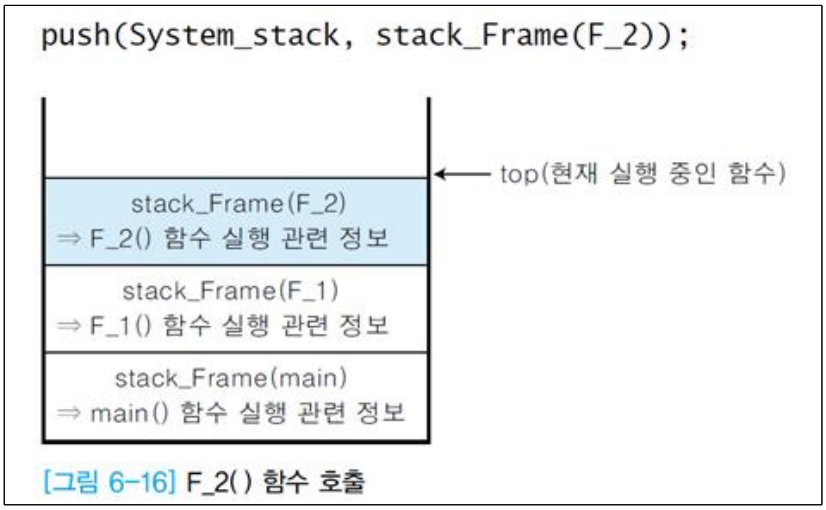
② mail() 함수 실행중에 F_1() 함수 호출을 만나면 함수 호출과 복귀 등 작업 전환을 위해 필요한 정보를 스택 프레임에 저장하여 시스템 스택에 삽입한다. 이때 스택 프레임에는 호출된 F_1() 함수의 수행이 끝나고 main()함수로 복귀할 주소 a가 저장된다



[그림 6-15] F_1() 함수 호출

③ 호출된 함수 F_1() 함수를 실행

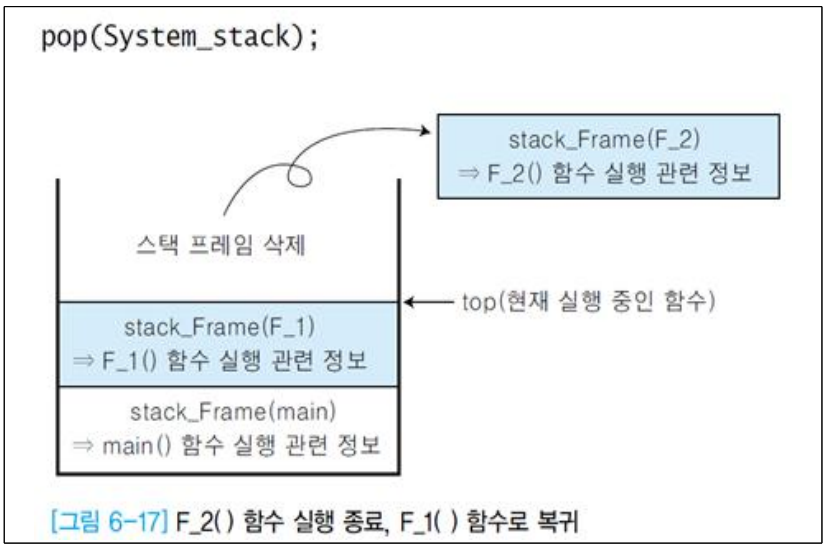
④ F_1() 함수 실행중에 F_2()함수 호출을 만나면 함수 호출과 복귀 등 작업 전환을 위해 필요한 정보를 새로운 스택 프레임에 저장하여 시스템 스택에 삽입한다. 이때 스택 프레임에는 F_1() 함수로 복귀할 주소 b가 저장된다



[그림 6-16] F_2() 함수 호출

⑤ 호출된 함수 F_2() 함수를 실행한다

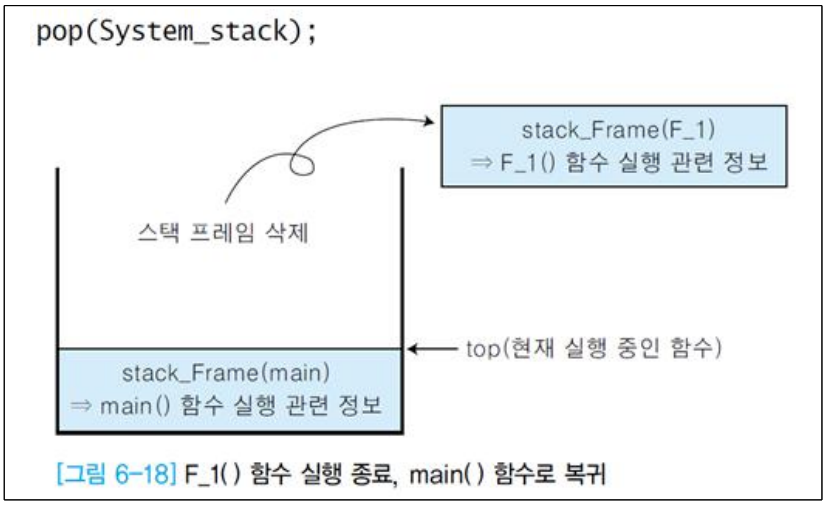
⑥ F_2() 함수 실행이 끝나면 F_2() 함수를 호출했던 이전 위치로 복귀하여 이전 함수 F_1()의 작업을 계속해야 한다. 그러려면 이전 작업에 관련한 지역변수, 매개 변수 및 복귀 주소 등의 정보가 필요한데, 이런 정보들은 시스템 스택에 있으므로 시스템 스택의 top에 있는 스택 프레임을 pop하여 정보를 확인하고 복귀 및 작업 전환을 실행



[그림 6-17] F_2( ) 함수 실행 종료, F_1( ) 함수로 복귀

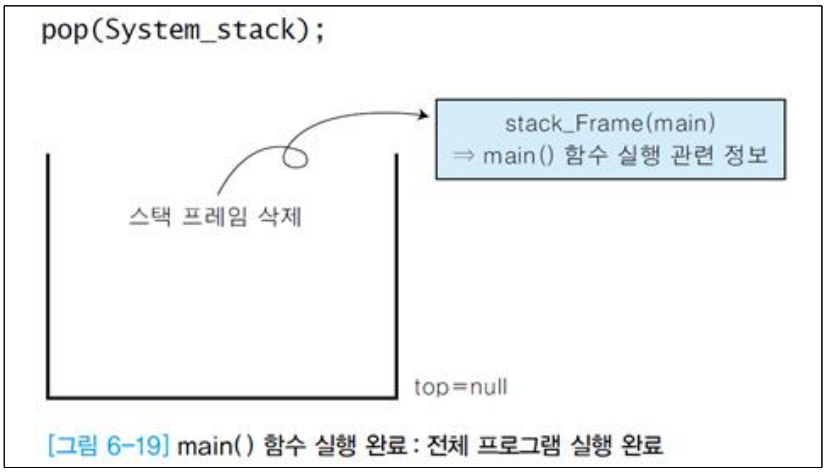
⑦ F_1()함수로 복귀하여 F_1() 함수의 나머지 부분을 실행한다

⑧ F_1() 함수 실행이 끝났으므로 F_1() 함수를 호출했던 main() 함수의 이전 위치로 다시 복귀하여 실행을 계속해야 한다. 그러려면 복귀할 주소 등의 작업 전환 정보가 필요한데 이런 정보들을 얻기 위해서 시스템 스택의 top에 있는 스택 프레임을 pop 하여 정보를 확인하고 복귀 및 작업 전환을 실행한다



[그림 6-18] F_1( ) 함수 실행 종료, main( ) 함수로 복귀

⑨ main() 함수로 복귀하여 나머지 부분을 실행하고 실행이 끝나면 다시 시스템 스택의 top에 있는 스택 프레임을 pop하여 복귀를 실행한다. 주 프로그램인 main() 함수는 호출을 한 상위 함수, 즉 복귀할 이전 함수의 주소를 가지고 있지 않으므로 전체 프로그램 수행이 완료된다. 정상적인 함수 호출과 복귀가 모두 완료되었으므로 시스템 스택은 공백 스택이 된다




[그림 6-19] main() 함수 실행 완료 : 전체 프로그램 실행 완료

## 학습내용2 : 수식의 괄호 검사

### 1. 수식의 괄호 검사

* 수식에 포함되어 있는 괄호는 가장 마지막에 열림 괄호를 가장 먼저 닫아 주어야 하는 후입선출 구조로 구성되어있으므로 후입선출 구조의 스택을 이용하여 괄호를 검사한다

* 수식을 왼쪽에서 오른쪽으로 하나씩 읽으면서 괄호 검사
- 왼쪽 괄호를 만나면 스택에 push
- 오른쪽 괄호를 마나면 스택을 pop하여 마지막에 저장한 괄호와 같은 종류인지를 확인
(같은 종류의 괄호가 아닌 경우 괄호의 짝이 잘못 사용된 수식임)

* 수식에 대한 검사가 모두 끝났을 때 스택은 공백 스택이 됨
- 수식이 끝났어도 스택이 공백이 되지 않으면 괄호의 개수가 틀린 수식임

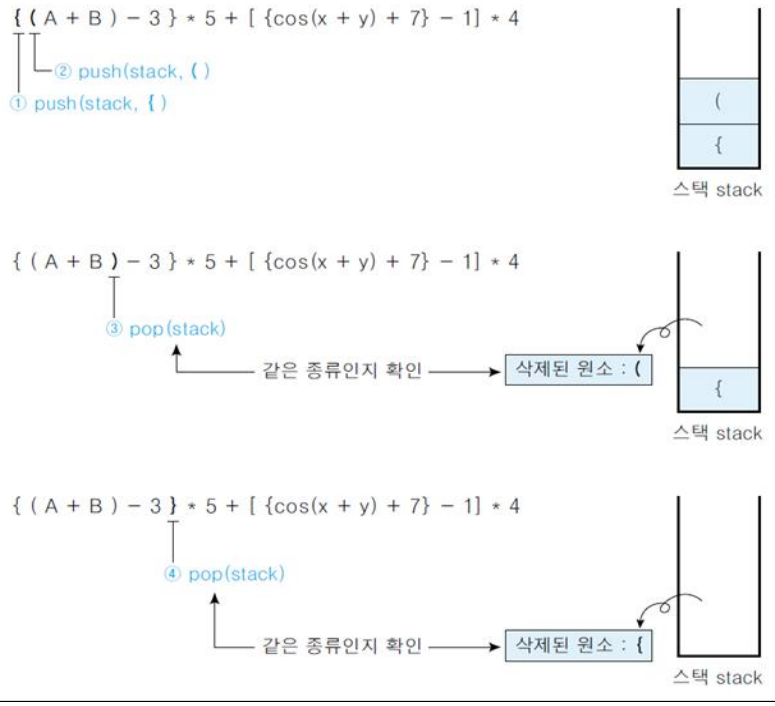
## 2. 괄호의 쌍 검사 알고리즘

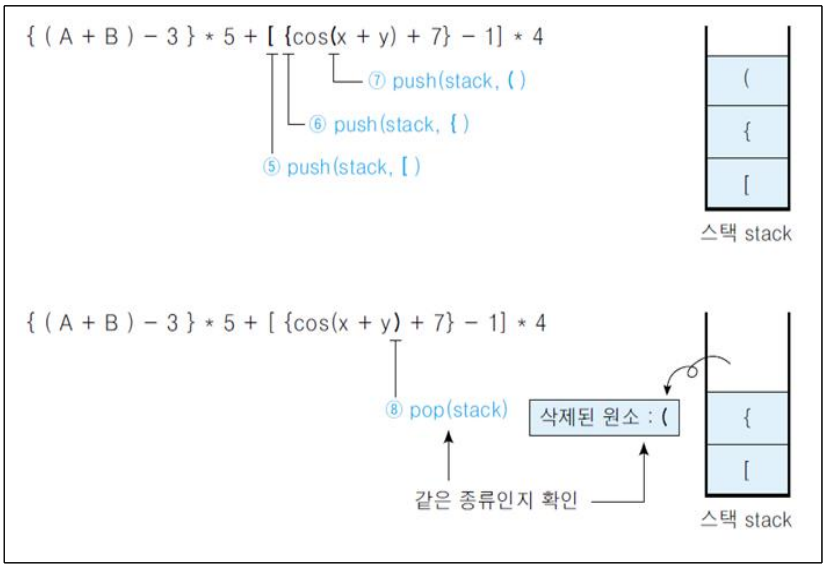
**알고리즘 6-3** 괄호의 쌍 검사 알고리즘

```
testPair()
    exp ← Expression;
    Stack ← null;
    while (true) do {
        symbol ← getSymbol(exp);
        case {
            symbol = "(" or "[" or "{" :
                push(Stack, symbol);
            symbol = ")" :
                open_pair ← pop(Stack);
                if (open_pair ≠ "(") then return false;
            symbol = "]" :
                open_pair ← pop(Stack);
                if (open_pair ≠ "[") then return false;
            symbol = "}" :
                open_pair ← pop(Stack);
                if (open_pair ≠"{") then return false;
            symbol = null :
                if (isEmpty(Stack)) then return true;
                else return false;
            else :
        }
    }
end testPair()
```

## 3. 예

{(A+B)−3}*5+[{cos(x+y)+7}−1]*4





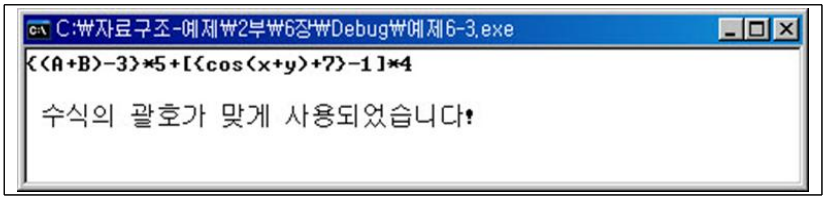

## 4. 수식의 괄호 검사 C 프로그램

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef char element;

typedef struct stackNode {
    element data;
    struct stackNode *link;
}stackNode;

stackNode* top;

void push(element item)
{
    stackNode* temp=(stackNode *)malloc(sizeof(stackNode));
    temp->data = item;
    temp->link = top;
    top = temp;
}

element pop()
{
    element item;
    stackNode* temp=top;

    if(top == NULL) {
        printf("₩n₩n Stack is empty !₩n");
        return 0;
    }
    else {
        item = temp->data;
        top = temp->link;
        free(temp);
        return item;
    }
}

element peek()
{
    element item;
    if(top == NULL) {
        printf("₩n₩n Stack is empty !₩n");
        return 0;
    }
    else {
        item = top->data;
        return item;
    }
}

void del()
{
    stackNode* temp;
```

```
055     if(top == NULL) {
056         printf("₩n₩n Stack is empty !₩n");
057
058     }
059     else {
060         temp = top;
061         top = top->link;
062         free(temp);
063     }
064 }
065
066 void printStack()
067 {
068     stackNode* p=top;
069     printf("₩n STACK [ ");
070     while(p){
071         printf("%d ",p->data);
072         p = p->link;
073     }
074     printf("] ");
075 }
076
077 int testPair(char *exp)
078 {
079     char symbol, open_pair;
080     int i, length=strlen(exp);
081     top=NULL;
082
```

```
108
109 void main(void)
110 {
111     char* express = "{(A+B)-3}*5+[{cos(x+y)+7}-1]*4";
112     printf("%s", express);
113     if(testPair(express) == 1)
114         printf("₩n₩n 수식의 괄호가 맞게 사용되었습니다!");
115     else
116         printf("₩n₩n 수식의 괄호가 틀렸습니다!");
117
118     getchar();
119 }
```

* 실행결과




## 학습내용3 : 수식의 후위표기와 계산

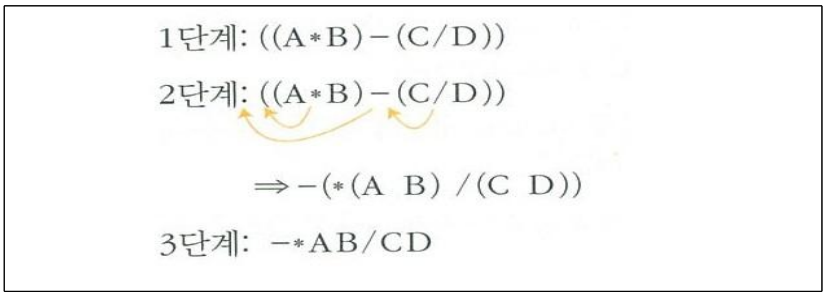
### 1. 수식을 표현하는 방법

* 전위 표기법 : 연산자를 피연산자 앞에 표기하는 방법 예) +AB
* 중위 표기법 : 연산자를 피연산자의 가운데에 표기하는 방법이다 예) A+B
* 후위 표기법 : 연산자를 피연산자의 뒤에 표기하는 방법 예) AB+

### 2. 중위 표기식의 전위 표기식으로 변환 방법

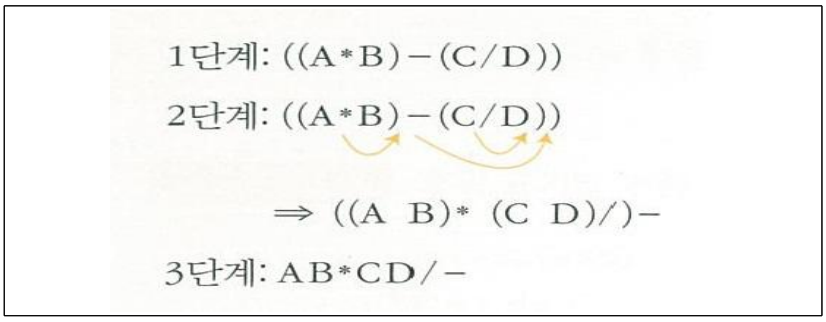
① 수식의 각 연산자에 대해서 우선순위에 따라 괄호를 사용하여 다시 표현한다
② 각 연산자를 그에 대응하는 왼쪽 괄호의 앞으로 이동시킨다
③ 괄호를 제거한다

* 예) A*B-C/D




### 3. 중위 표기식의 후위 표기식으로 변환 방법

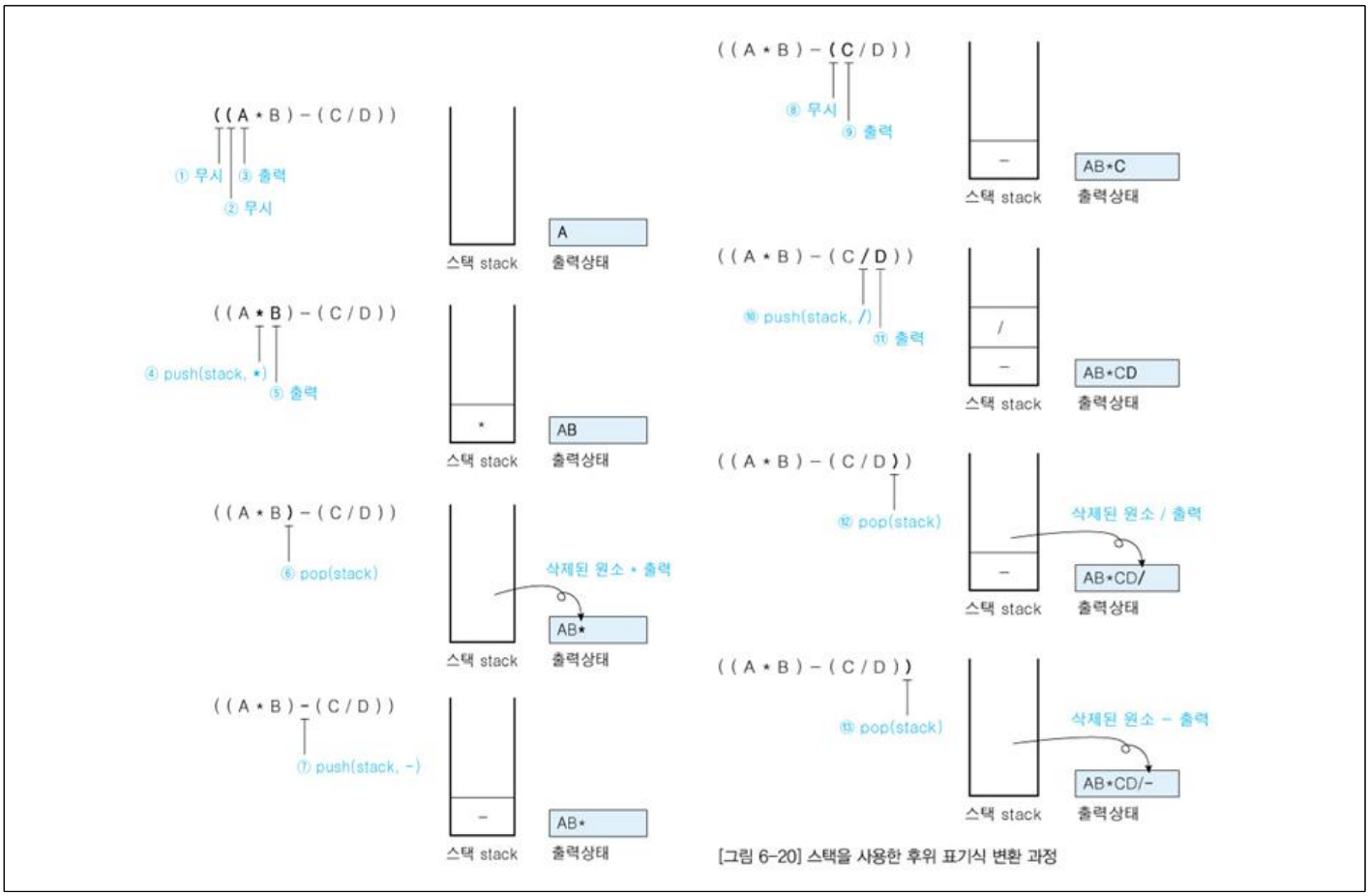
① 수식의 각 연산자에 대해서 우선순위에 따라 괄호를 사용하여 다시 표현한다
② 각 연산자를 그에 대응하는 오른쪽 괄호의 뒤로 이동시킨다
③ 괄호를 제거한다




### 4. 스택을 사용하여 중위 표기식을 후위 표기식으로 변환

① 왼쪽 괄호를 만나면 무시하고 다음 문자를 읽는다
② 피연산자를 만나면 출력한다
③ 연산자를 만나면 스택에 push한다
④ 오른쪽 괄호를 만나면 스택을 pop하여 출력한다
⑤ 수식이 끝나면 스택이 공백이 될 때까지 pop하여 출력한다

* 예) ((A*B)-(C/D))




[그림 6-20] 스택을 사용한 후위 표기식 변환 과정

## 5. 후위 표기법 변환 알고리즘

**알고리즘 6-4** 후위 표기법 변환 알고리즘

```
infix_to_postfix(exp)
    while(true) do {
        symbol ← getSymbol(exp);
        case {
            symbol = operand :              // 피연산자 처리
                print(symbol);
            symbol = operator :             // 연산자 처리
                push(stack, symbol);
            symbol = ")" :                  // 오른쪽 괄호 처리
                print(pop(stack));
            symbol = null :                 // 수식의 끝 처리
                while(top > -1) do
                    print(pop(stack));
            else :
        }
    }
end  infix_to_postfix( )
```

## 6. 스택을 사용하여 후위 표기식을 연산

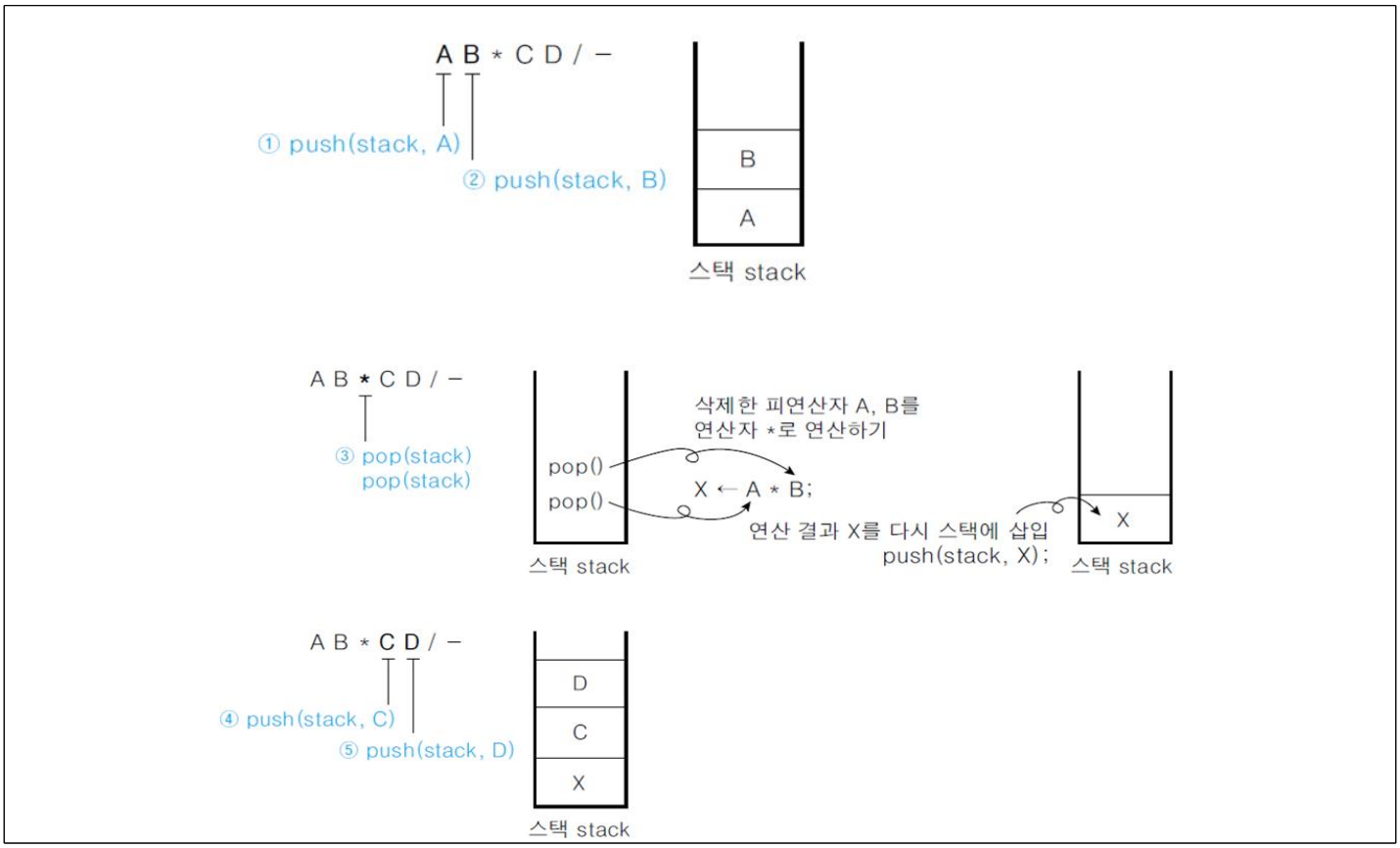
① 피연산자를 만나면 스택에 push한다
② 연산자를 만나면 필요한 만큼의 피연산자를 스택에서 pop하여 연산하고, 연산결과를 다시 스택에 push한다
③ 수식이 끝나면, 마지막으로 스택을 pop하여 출력한다
* 수식이 끝나고 스택에 마지막으로 남아있는 원소는 전체 수식의 연산결과값이 된다

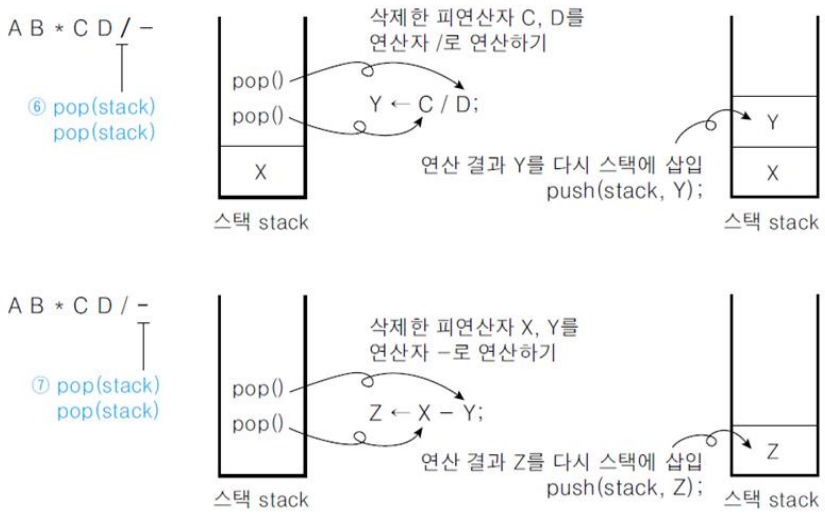
## 7. 후위 표기 수식의 연산 알고리즘

**알고리즘 6-5** 후위 표기 수식의 연산 알고리즘

```
evalPostfix(exp)
    while (true) do {
        symbol ← getSymbol(exp);
        case {
            symbol = operand :          // 피연산자 처리
                push(Stack, symbol);
            symbol = operator :         // 연산자 처리
                opr2 ← pop(Stack);
                opr1 ← pop(Stack);
                result ← opr1 op(symbol) opr2;
                // 스택에서 꺼낸 피연산자들을 연산자로 연산
                push(Stack, result);
            symbol = null :             // 후위수식의 끝
                print(pop(Stack));
        }
    }
end evalPostfix()
```

* 예) AB*CD/-






[그림 6-22] 스택을 사용한 후위 표기법 연산 과정

## 8. 후위 표기식의 연산 C 프로그램

```
001 #include <stdio.h>
002 #include <stdlib.h>
003 #include <string.h>
004
005 typedef int element;
006
007 typedef struct stackNode {
008     element data;
009     struct stackNode *link;
010 }stackNode;
011
012 stackNode* top;
013
014 void push(element item)
015 {
016     stackNode* temp=(stackNode *)malloc(sizeof(stackNode));
017     temp->data = item;
018     temp->link = top;
019     top = temp;
020 }
021
022 element pop()
023 {
024     element item;
025     stackNode* temp=top;
026
```

```
027     if(top == NULL) {
028         printf("₩n₩n Stack is empty !₩n");
029         return 0;
030     }
031     else {
032         item = temp->data;
033         top = temp->link;
034         free(temp);
035         return item;
036     }
037 }
038
039 element peek()
040 {
041     element item;
042     if(top == NULL) {
043         printf("₩n₩n Stack is empty !₩n");
044         return 0;
045     }
046     else {
047         item = top->data;
048         return item;
049     }
050 }
051
052 void del()
053 {
054     stackNode* temp;
```

```
055     if(top == NULL) {
056         printf("₩n₩n Stack is empty !₩n");
057
058     }
059     else {
060         temp = top;
061         top = top->link;
062         free(temp);
063     }
064 }
065
066 void printStack()
067 {
068     stackNode* p=top;
069     printf("₩n STACK [ ");
070     while(p){
071         printf("%d ",p->data);
072         p = p->link;
073     }
074     printf("] ");
075 }
076
077 element evalPostfix(char *exp)
078 {
079     int opr1, opr2, value, i=0;
080     int length = strlen(exp);
081     char symbol;
082     top = NULL;
```

* 실행결과


## 【학습정리】

1. 프로그램 간의 호출과 복귀에 따른 수행 순서는 가장 나중에 호출된 함수가 가장 먼저 실행을 완료하고 복귀하는 LIFO 구조이다. LIFO 구조를 갖는 스택을 사용하여 호출과 복귀 순서를 관리할 수 있는데 이러한 스택을 시스템 스택이라고 한다.

2. 괄호가 포함된 수식에서 괄호의 쌍이 제대로 사용되었는지를 검사하기 위해서 스택을 사용한다.