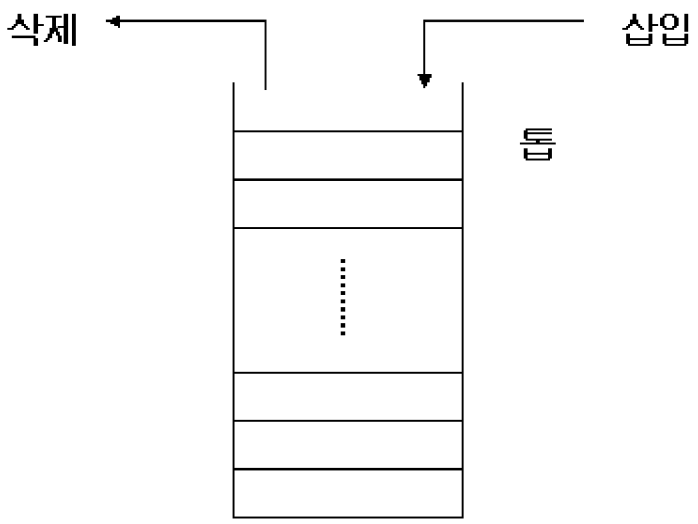
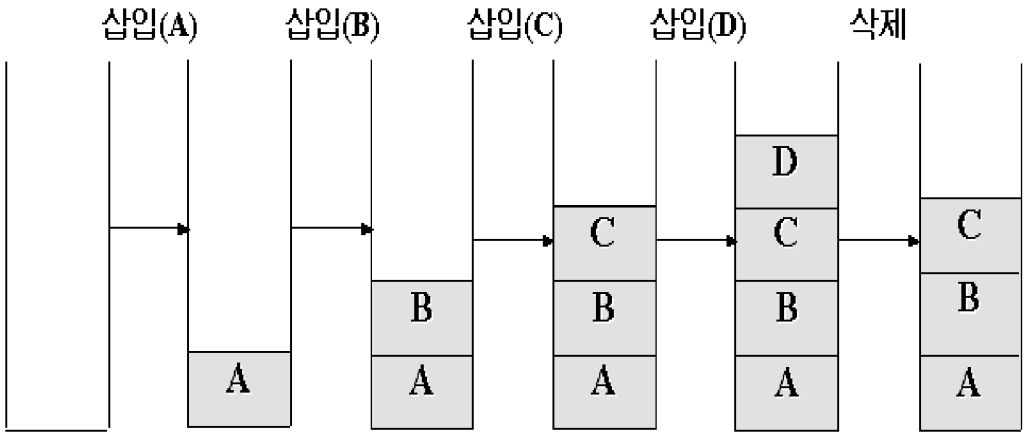
11. 스택 (stack)

스택의 정의 : 삽입과 삭제가 한쪽 끝 톱(top)에서만 이루어지는 유한 순서 리스트(Finite ordered list)



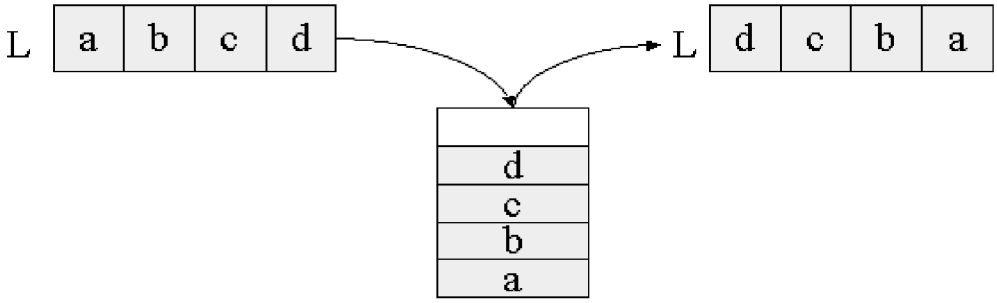
후입 선출(Last-In-First-Out (LIFO))

삽입 : Push, 삭제 : Pop
스택을 Pushdown 리스트라고도 함



스택의 응용

리스트의 순서를 역순으로 만드는 데 유용



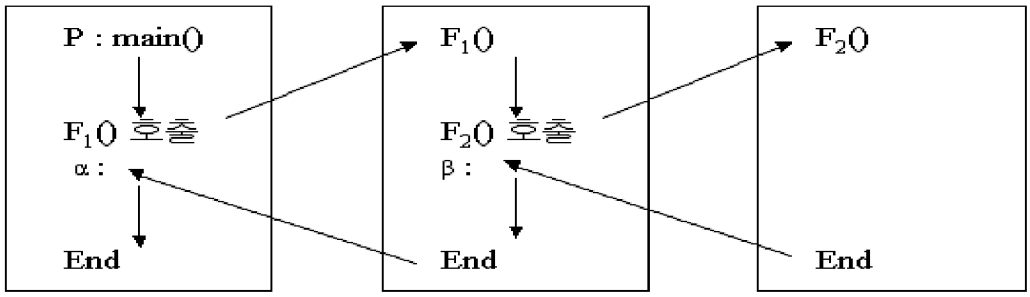
시스템 스택(system stack) 또는 실행시간스택(runtime stack)

시스템 스택

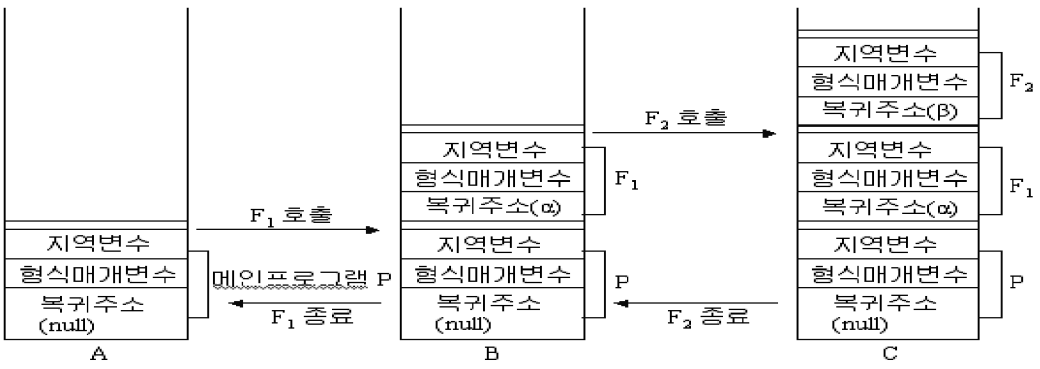
① 프로그램 간의 호출과 복귀에 따른 실행 순서 관리

② 활성화 레코드

- 복귀 주소, 형식 매개 변수, 지역 변수들을 포함
- 항상 스택의 톱에는 현재 실행되는 함수의 활성화 레코드 존재



시스템 스택의 변화



순환 호출(recursive call)

순환 호출

① 순환 호출이 일어날 때마다 활성화 레코드가 만들어져 시스템 스택에 삽입됨
② 가능한 호출의 횟수는 활성화 레코드의 개수를 얼마로 정하느냐에 따라 결정
③ 순환의 깊이가 너무 깊을 때
- 프로그램 실행 중단의 위험성

순환 프로그램의 실행이 느린 이유
활성화 레코드들의 생성과 필요한 정보 설정 등의 실행 환경 구성에 많은 시간 소요

스택 추상 데이타 타입

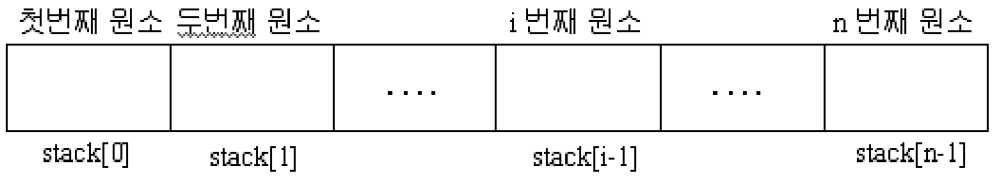
스택 ADT

```
ADT Stack
      데이타 :  0개 이상의 원소를 가진 유한 순서 리스트
      연산 :  stack ∈ Stack; item ∈ Element;
            createStack() ::= create an empty stack;
            push(stack, item) ::= insert item onto the top of Stack;
            isEmpty(stack) ::= if (stack is empty) then return true
                        else return false;
            pop(stack) ::= if isEmpty(stack) then return error
                        else { delete and return the top item of stack };
            delete(stack) ::= if isEmpty(stack) then return error
                        else delete the top item;
            peek(stack) ::= if (isEmpty(stack)) then return error
                        else return { the top item of the stack };
End Stack
```

스택의 순차 표현

1차원 배열, stack[n]을 이용한 순차 표현

① 스택을 표현하는 가장 간단한 방법
② n 은 스택에 저장할 수 있는 최대 원소 수
③ 스택의 i 번째 원소는 stack[i-1] 에 저장
④ 변수 top 은 스택의 톱 원소를 가리킴 (초기: top = -1)



스택의 순차 표현 연산자

createStack과 isEmpty 연산자의 구현

```
createStack()   // 공백 순차 스택을 생성
    stack[n];  // 인덱스는 0에서 n-1
    top ← -1;
end createStack()

isEmpty(stack)   // 스택이 공백인가를 검사
    if (top < 0) then return true
    else return false;
end isEmpty()
```

push, pop 연산자의 구현

```
push(stack, item)              // 스택 stack의 톱에 item을 삽입
        if (top >= n-1) then stackFull();      // stackFull()은 현재의 배열에
        top ← top +1;          // 원소가 만원이 되었을 때 배열을
        stack[top] ← item;     // 확장하는 함수
end push()
pop(stack)                     // 스택 stack의 톱 원소를 삭제하고 반환
         if (top < 0) then stackEmpty()
                               // stackEmpty()는 stack이 공백인 상태를 처리
         else {
                  item ← stack[top];
                  top ← top -  1;
                  return item;
         }
end pop()
```

delete, peek 연산자의 구현

```
delete(stack)  // 스택의 톱 원소를 삭제
        if (top < 0) then stackEmpty()      // stackEmpty()는 stack이
        else top ←  top -  1;               // 공백인 상태를 처리
         end delete()
peek(stack)    // 스택의 톱 원소를 검색
        if (top < 0) then stackEmpty()      // stackEmpty()는 stack이
        else return stack[top];             // 공백인 상태를 처리
end peek()
```

C 에서 스택의 구현

스택 ADT를 C로 구현

스택 structure 의 정의

```
typedef struct {     /* 스택의 원소 타입 */
            int id;
            char name[10];
            char grade;
} element;
```

스택 structure 의 선언

```
element stack[STACK_SIZE];              /* 배열로 표현된 스택 */
int top = -1;                           /* top을 초기화 */
```

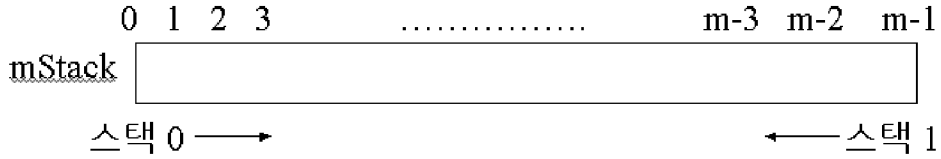
배열로 stack 을 구현한 C 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_SIZE 100              /* 스택의 최대 크기 */
typedef int element;            /* element를 int 타입으로 정의*/
element stack[STACK_SIZE];

void push(int *top, element item) {
   if(*top >= STACK_SIZE - 1) { /* 스택이 만원인 경우 */
        printf(" Stack is full\n");
        return;
    }
    stack[++(*top)] = item;          /* top은 top+1로*/
}

element pop(int *top) {
  if (*top == -1) {           /* 스택이 공백인 경우 */
            printf("Stack is empty\n");
  exit(1);
 }
 else return stack[(*top)--];  /* top은 top-1로 */
}
```

```
int isEmpty(int *top) {
  if (*top == -1) return 1;            /* 공백이면 1, 공백이 아니면 0 */
       else return 0;
}
void delete(int *top) {
  if (*top == -1) {            /* 스택이 공백인 경우 */
              printf("Stack is empty\n");
              exit(1);
  }
  else (*top)--;
}

element peek(int top) {
   if (top == -1) {젩           /* 스택이 공백인 경우 */
                printf("Stack is empty\n");
                exit(1);
  }
  else return stack[top];
}
int main( void ) {
        int top=-1;
        element data1, data2;
        printf("push data1 : %d\n", 1);
        push(&top, 1);
        printf("push data2 : %d\n", 2);
        push(&top, 2);
        data2 = peek(top);
        printf("peek data2 : %d\n", data2);
        delete(&top);
        printf("delete data2\n");
        data1 = pop(&top);
        printf("pop data1 : %d\n", data1);
        return 0;
```

복수 스택의 순차 표현

① 하나의 배열(순차 사상)을 이용하여 여러 개의 스택을 동시에 표현하는 방법
② 두 개의 스택인 경우
- 스택 0은 mStack[m-1]쪽으로
- 스택 1은 mStack[0] 쪽으로 확장시키면 됨

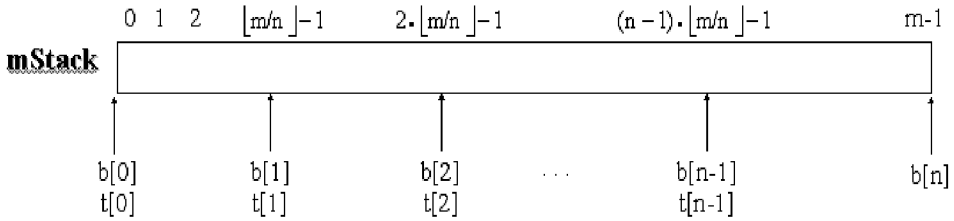


n개의 스택인 경우

① 각 스택이 n개로 분할된 메모리 세그먼트 하나씩 할당
② n개의 스택에 균등 분할된 세그먼트 하나씩 할당한 뒤의 초기 배열 mStack[m]

b[i] = t[i] = i · [m/n]-1

b[i] / t[i] : 스택 i (0 <= i <= n-1)의 최하단 / 최상단 원소



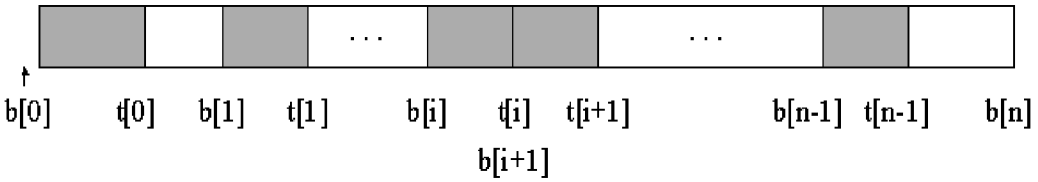
복수 스택을 위한 스택 연산 (1)

```
isEmpty(i)        // 스택 i의 공백 검사
        if (t[i] = b[i]) then return true
        else  return false;
end isEmpty()

push(i, item)       // 스택 i에 item을 삽입
        if (t[i] = b[i+1]) then stackFull(i);       // 스택 확장
        t[i] ← t[i] + 1;
        mStack[t[i]] ← item;
end push()
```

```
pop(i)      // 스택 i에서 톱 원소를 삭제하여 반환
        if (t[i] = b[i]) then return null
        else item ← mStack[t[i]];
        t[i] ← t[i] - 1;
        return item;
end pop()

delete(i)       // 스택 i에서 톱 원소를 삭제
        if (t[i] = b[i]) then return
        else t[i] ← t[i] - 1;
end delete()

peek(i)       // 스택 i에서 톱 원소를 검색
        if (t[i] = b[i]) then return null
        else return mStack[t[i]];
end peek()
```

stackFull 알고리즘

문제점 : 스택 i는 만원이지만 배열 mStack에는 자유 공간이 있는 경우가 발생



해결책 : 배열 mStack에 가용공간이 남아있는지 찾아보고 있으면 스택들을 이동시켜 가용 공간을 스택 i가 사용할 수 있도록 함

stackFull 알고리즘의 구현

알고리즘

```
// 스택 i의 오른편에서 가용공간을 가진 스택을 찾는다.
if  ((i < j <= n-1) and (t(j) < b(j+1))인 스택 j가 있으면)
      then 스택 i+1, i+2, ..., j를 오른쪽으로 한자리 이동
// 스택 i의 왼편에서 가용공간을 가진 스택을 찾는다.
else if  ((0 <= j < i) and (t(j) < b(j+1)) 인 스택 j가 있으면)
then 스택 j+1, j+2, ..., i를 왼쪽으로 한자리 이동

// 두 경우 모두 실패하면, mStack에 가용 공간이 없으므로
else  오버플로우
```

문제점

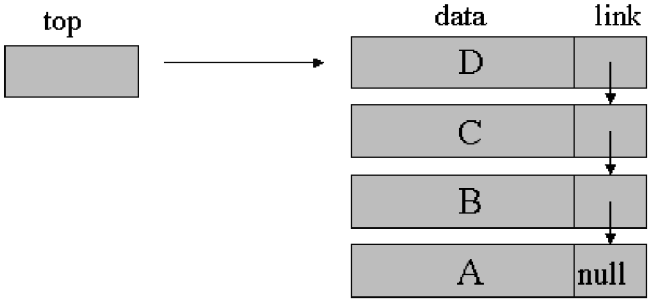
- 최악의 경우 원소 삽입시 항상 스택의 이동 발생
- 해결책 : 비순차 표현으로 구현

연결리스트로 표현된 연결 스택(linked stack)

① top 이 지시하는 연결리스트로 표현
- 스택의 변수 top은 톱 원소 즉 첫 번째 원소를 가리킴

② 원소의 삽입
- 생성된 노드를 연결리스트의 첫 번째 노드로 삽입

③ 원소의 삭제
- 항상 연결리스트의 첫 번째 노드를 삭제하고 원소값을 반환



연결 스택의 연산자

```
createStack()
    // 공백 연결 스택 생성
    top ← null
end createStack()

isEmpty(stack)
    // 연결 스택의 공백 검사
    if (top = null) then return true
    else return false;
end isEmpty()

push(stack, item)
     // 연결 스택 톱에 item을 삽입
     newNode ← getNode();
     newNode.data ← item;
     newNode.link ← top;
     top ← newNode
end push()
```

```
pop(stack)
    // 연결 스택에서 톱 원소를 삭제하여 반환
    if (top = null) then return null
    else {
        item ← top.data;
        oldNode ← top;
        top ← top.link;
        retNode(oldNode);
        return item;
    }
end pop()
delete(stack)
    // 연결 스택에서 톱 원소를 삭제
    if (top = null) then return
    else {
        oldNode ← top;
        top ← top.link;
        retNode(oldNode);
    }
end delete()

peek(stack)
    // 스택의 톱 원소를 검색
    if (top = null) then return null
    else return (top.data);
end peek()
```

k개의 스택의 연결 표현

stackTop[k] : 스택의 톱(top) 을 관리하는 배열

연산

```
isEmpty(i)  //스택 i의 공백 검사
        if (stackTop[i] = null) then return true
        else return false;
end isEmpty()
```

```
push(i, item)  // 스택 i에 item을 삽입
        newNode ← getNode();
        newNode.data ← item;
        newNode.link ← stackTop[i];
        stackTop[i] ← newNode
end push()
pop(i)   // 스택 i에서 원소를 삭제하고 반환
        if (stackTop[i] = null) then return null
        else {
              item ← stackTop[i].data;
              oldNode ← stackTop[i];
                    stackTop[i] ← stackTop[i].link;
              retNode(oldNode);
              return item;
        }
end pop()

delete(i)  //스택 i에서 원소를 삭제
        if (stackTop[i] = null) then return
        else {
              oldNode ← stackTop[i];
              StackTop[i] ← stackTop[i].link;
              retNode(oldNode);
        }
end delete()

peek(i)  //스택 i에서 원소 검색
        if (stackTop[i] = null) then return null
        else return (stackTop[i].data);
end peek()
```

C 리스트를 이용한 스택 구현

스택의 원소를 저장할 노드의 표현

- 스택에서 처리해야 되는 원소의 구조

```
typedef struct {              /* 스택 원소 구조 */
        int id;
        char name[10];
        char grade;
} element;
```

- 원소를 저장할 연결 노드의 구조와 top의 정의

```
typedef struct stackNode{  /* 리스트 노드 구조 */
        element data;
        struct stackNode *link;
} stackNode
stackNode *top = NULL;  /* 공백 연결 스택 top을 생성 */
```

연결 리스트로 Stack을 구현한 C 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {                    /*스택 원소 구조 */
        int id;
        char name[10];
        char grade;
}element ;
typedef struct stackNode {          /* 리스트 노드 구조 */
   element data;
   struct stackNode *link;
}stackNode ;

void push(stackNode **top, element data) {
   /* 스택의 톱에 원소를 삽입 */
     stackNode* temp;
    temp = (stackNode*)malloc(sizeof(stackNode));
    temp->data = data;
    temp->link = *top;
    *top = temp;
}
```

```
 element pop(stackNode **top) {
       /*스택의 톱 원소를 반환하고 노드는 삭제 */
      stackNode* temp;
      element data;
      temp = *top;
      if(temp == NULL) {           /* 스택이 공백이면 */
       printf("Stack is empty\n");
        exit(1);
      }
      else {
        data = temp->data;
        *top = temp->link;
         free(temp);                    /* 연결 리스트에서 노드를 삭제 */
         return data;
      }
}
 element peek(stackNode *top) {    /*스택의 톱 원소를 검색 */
      element data;
      if(top == NULL) {              /* 스택이 공백이면 */
           printf("Stack is empty\n");
           exit(1);
      }
      else {
            data = top->data;
            return data;
      }
}

 void delete(stackNode **top) {     /*스택의 톱 원소를 삭제 */
    stackNode* temp;
     if(*top == NULL) {            /* 스택이 공백이면 */
      printf("Stack is empty\n");
      exit(1);
    }
    else {
       temp = *top;
       *top = (*top)->link;
        free(temp);
    }
}
```

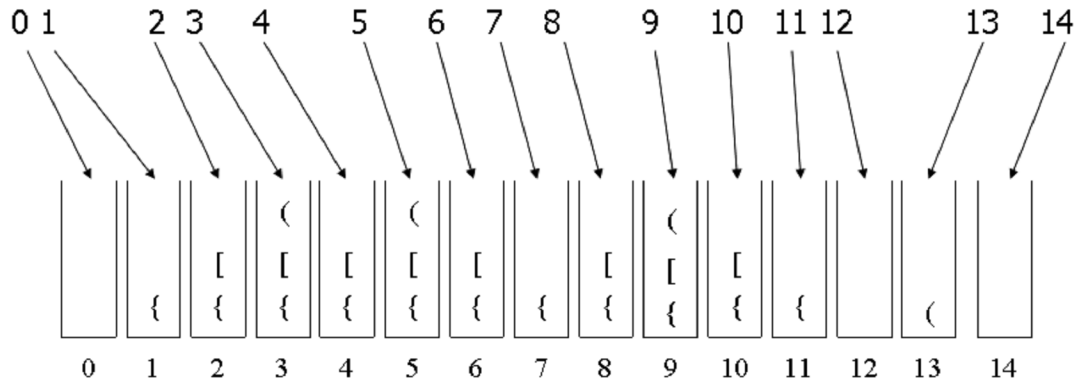
```
int main( void ) {
  stackNode *top = NULL;            /*공백 연결 스택으로 top을 선언 */
  element data1, data2, data3, data4;
  data1.id = 1;
  strcpy(data1.name, "Lee");
  data1.grade = 'A';
  data2.id = 2;
  strcpy(data2.name, "Park");
  data2.grade = 'B';
  printf("push data1 : (%d, %s, %c)\n", data1.id, data1.name, data1.grade);
  push(&top, data1);
  printf("push data2 : (%d, %s, %c)\n", data2.id, data2.name, data2.grade);
  push(&top, data2);
  data3 = peek(top);
  printf("peek data2 : (%d, %s, %c)\n", data3.id, data3.name, data3.grade);
  delete(&top);
  printf("delete data2\n");
  data4 = pop(&top);
  printf("pop data1 : (%d, %s, %c)\n", data4.id, data4.name, data4.grade);
  return 0;
}
```

수식의 괄호쌍 검사

수식 : 대괄호, 중괄호, 소괄호를 포함
스택을 이용한 수식의 괄호 검사 알고리즘

```
parenTest( ) {  // 괄호가 올바로 사용되었으면 true를 반환
      exp ← Expression;   //수식의 끝은 ∞문자로 가정
      parenStack ← null;
      while true do {
            symbol ← getSymbol(exp);
            case {
              symbol = "(" or "[" or "{":  push(parenStack, symbol);
              symbol = ")": left ← pop(parenStack);
                           if (left ≠ "(") then return false;
              symbol = "]": left ← pop(parenStack);
                           if (left ≠ "[") then return false;
              symbol = " }": left ← pop(parenStack);
                           if (left ≠ "{") then return false;
              symbol = "∞": if (isEmpty(parenStack)) then return true
                                    else return false;
                else:   // 괄호 이외의 수식 문자
            } //end case
      } //end while
end parenTest()
```

{ a2 - [(b+c) 2 - (d+e) 2] * [sin (x - y)] } - cos (x+y)



스택을 이용한 수식의 계산

수식

① 연산자와 피연산자로 구성

② 피연산자

- 변수나 상수
- 피연산자의 값들은 그 위에 동작하는 연산자와 일치해야 함

③ 연산자

- 연산자들 간의 실행에 대한 우선 순위 존재
- 값의 타입에 따른 연산자의 분류

  기본 산술 연산자 : +, -, *, /

  비교연산자 : <, <=, >, >=, =, !=

  논리 연산자 : and, or, not 등

④ 예

A + B*C -D/E

수식의 표기법

중위 표기법(infix notation)

- 연산자가 피연산자 가운데 위치

예) A+B*C-D/E

전위 표기법(prefix notation)

- 연산자가 피연산자 앞에 위치

예) -+A*BC/DE

후위 표기법(postfix notation)

① 연산자가 피연산자 뒤에 위치

② 폴리쉬 표기법(polish notation)

③ 예) ABC*+DE/-

④ 장점 : 연산 순서가 간단 - 왼편에서 오른편으로 연산자 기술 순서대로 계산

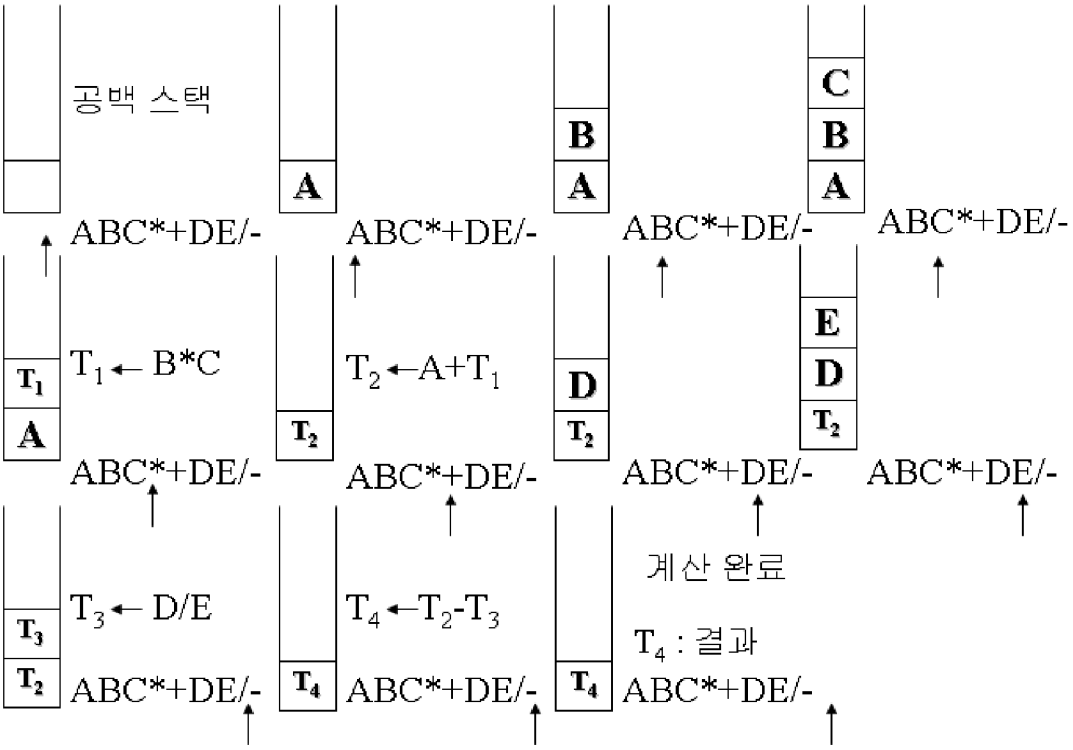
괄호가 불필요

후위 표기식(ABC*+DE/-)의 계산

후위 표기식	연산
ABC*+DE/-	$T_1 \leftarrow B*C$
AT_1+DE/-	$T_2 \leftarrow A+T_1$
T_2DE/-	$T_3 \leftarrow D/E$
T_2T_3-	$T_4 \leftarrow T_2-T_3$
T_4	

스택을 이용한 후위 표기식의 계산

후위 표기식 : ABC*+DE/-



후위 표기식 계산 알고리즘

```
evalPostfix(exp)   // 후위 표기식의 계산
     // 후위 표기식의 끝은 ∞이라고 가정
     // getToken은 식에서 토큰을 읽어오는 함수
     Stack[n]; // 피연산자를 저장하기 위한 스택
     top ← -1;
     while true do {
          token ← getToken(exp);
          case {
                    //토큰이 피연산자인 경우
               token = operand : push(Stack, token);
                    //토큰이 연산자인 경우
               token = operator : Stack에서 피연산자를 가져와 계산을 하고
                                  결과를 Stack에 저장;
            else : print(pop(Stack)); // 토큰이 ∞인 경우
          }
    }
end evalPostfix()
```

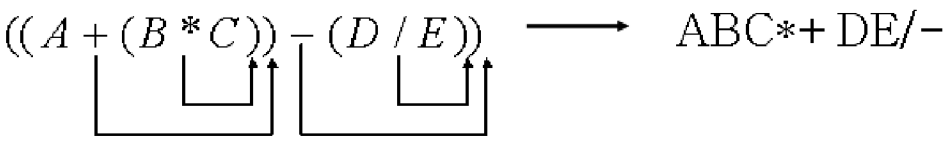
중위 표기식의 후위 표기식 변환

수동 변환 방법

1. 중위 표기식을 완전하게 괄호로 묶는다.
2. 각 연산자를 묶고 있는 괄호의 오른쪽 괄호로 연산자를 이동시킨다.
3. 괄호를 모두 제거한다.

피연산자의 순서는 불변

예 1



스택을 이용한 후위 표기식 변환 예

입력(중위 표기식)	토큰	스택	출력(후위 표기식)
A+B*C ∞			
A+B*C ∞	A		A
A+B*C ∞	+	+	A
A+B*C ∞	B	+	AB
A+B*C ∞	*	* +	AB
A+B*C ∞	C	* +	ABC
A+B*C ∞	∞		ABC*+

입력(중위 표기식)	토큰	스택	출력(후위 표기식)
A*(B+C)/D ∞			
A*(B+C)/D ∞	A		A
A*(B+C)/D ∞	*	*	A
A*(B+C)/D ∞	(	(*	A
A*(B+C)/D ∞	B	(*	AB
A*(B+C)/D ∞	+	+ (*	AB

입력(중위 표기식)	토큰	스택	출력(후위 표기식)
A*(B+C)/D ∞	C	+ (*	ABC
A*(B+C)/D ∞	)	*	ABC+
A*(B+C)/D ∞	/	/	ABC+*
A*(B+C)/D ∞	D	/	ABC+*D
A*(B+C)/D ∞	∞		ABC+*D/

후위 표기식 변환을 위한 우선 순위

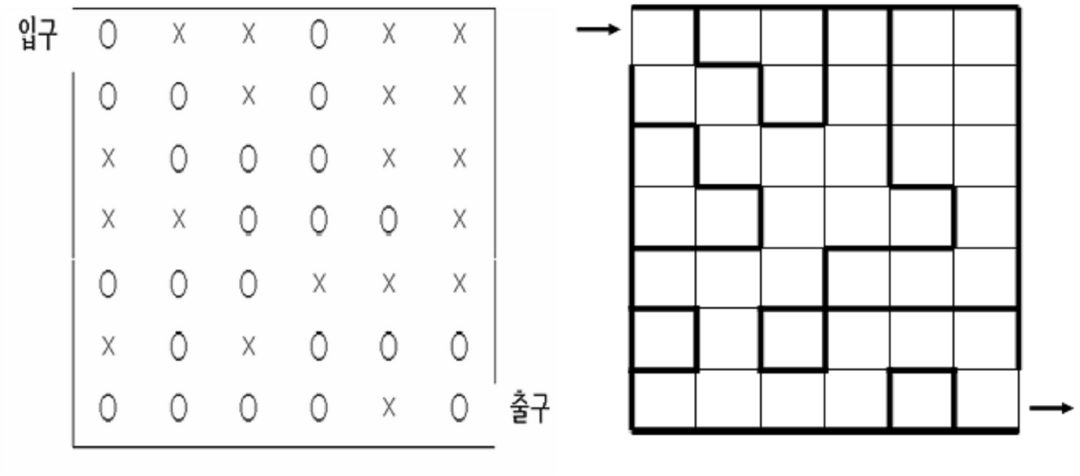
연산자	PIS (스택내 우선순위)	PIE (수식내 우선순위)
)	-	-
^	3	3
*,/	2	2
+,-	1	1
(	0	4

후위 표기식으로의 변환 알고리즘

```
makePostfix(e)
      // e는 주어진 중위표기식으로 끝은 ∞으로 표시
      // PIS와 PIE는 우선 순위를 반환해주는 함수
     // PIS (-∞) ← -1, stack[0] ← - ∞ , top ← 0, stack[n]을 가정
      while true do
            token ← getToken(e);
            case          {
             token = operand : print(token);
             token = ")" :
                  while stack[top] != "(" do print(pop(stack));
                  top ← top - 1;       // "("를 제거
             token = operator :         // "("가 제일 높은 PIE를 가짐.
                  while PIS(stack[top]) >= PIE(token) do print(pop(stack));
                  push(stack, token);
             token = ∞ :       //중위식의 끝
                  while top > -1 do print(pop(stack))
            } //end case
      } //end while
      print(' ∞');
      return;
end makePostfix()
```

미로 문제

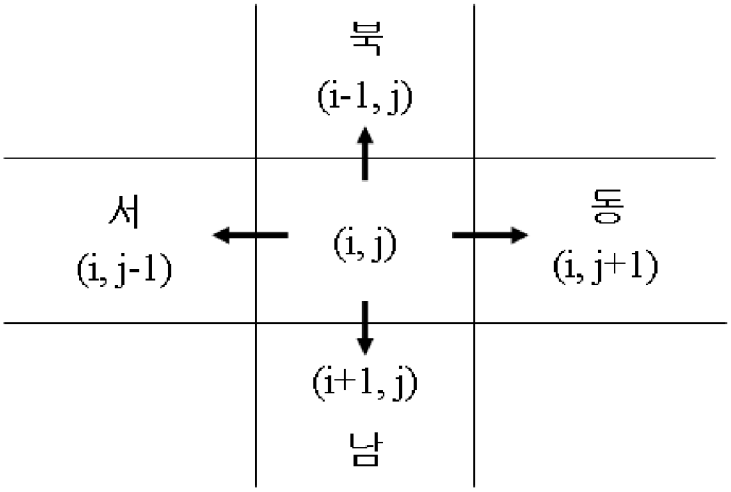
① 예)



② m X n 미로를 maze(m+2, n+2) 배열로 표현
- 사방을 1로 둘러싸서 경계 위치에 있을 때의 예외성(두 방향만 존재)을 제거

③ 현재 위치 : maze[i][j]

④ 이동 방향
- 북, 동, 남, 서 순서 (시계 방향)



⑤ 이동 방향 배열 : move[4, 2]

(dir)	row [0]	col [1]
북[0]	-1	0
동[1]	0	1
남[2]	1	0
서[3]	0	-1

⑥ 다음 위치계산 : maze[nexti,nextj]
- nextI <- i + move[dir, row]
- nextJ <- j + move[dir, col]

⑦ 방문한 경로를 mark[m+2, n+2] 에 저장
- 한 번 시도했던 위치로는 다시 이동하지 않음

⑧ 지나온 경로의 기억 <i, j, dir> 을 스택에 저장
- 스택의 최대 크기 : m × n

미로 경로 발견 알고리즘

```
mazePath( )
    maze[m+2, n+2];  // m x n 크기의 미로 표현
    mark[m+2, n+2];  // 방문 위치를 표시할 배열
    // 3 원소쌍 <i, j, dir> (dir = 0, 1, 2, 3) 을 저장하는 stack을 초기화
    stack[m×n];      top ← -1;
    push(stack, <1, 1, 1>);  // 입구위치 (1,1), 방향은 동(1)으로 초기화

    while (not isEmpty(stack)) do {        // 스택의 공백 여부를 검사
        <i, j, dir> ← pop(stack);          // 스택의 톱 원소를 제거
        while dir <= 3 do {        // 시도해 볼 방향이 있는 한 계속 시도
            nextI ← i + move[dir, row]; // 다음 시도할 행을 설정
            nextJ ← j + move[dir, col];          // 다음 시도할 열을 설정
            if (nextI = m and nextJ = n)
                then { print(path in stack);  print(i, j);  print(m, n);
                       return; }               // 미로 경로 발견
```

```
        if (maze[nextI, nextJ] = 0 and      // 이동 가능 검사
                mark[nextI, nextJ] = 0)     // 시도해 보지 않은 위치인지 검사
        then { mark[nextI, nextJ] ← 1;
                push(stack, <i, j, dir>);   // 이동한 위치를 스택에 기록
               <i, j, dir> ← <nextI, nextJ, 0>;  }  // 다음 시도할 위치와 방향
        else dir ← dir + 1;                 // 다음 방향 설정
                }
            }
            print("no path found");
    end mazePath()
```