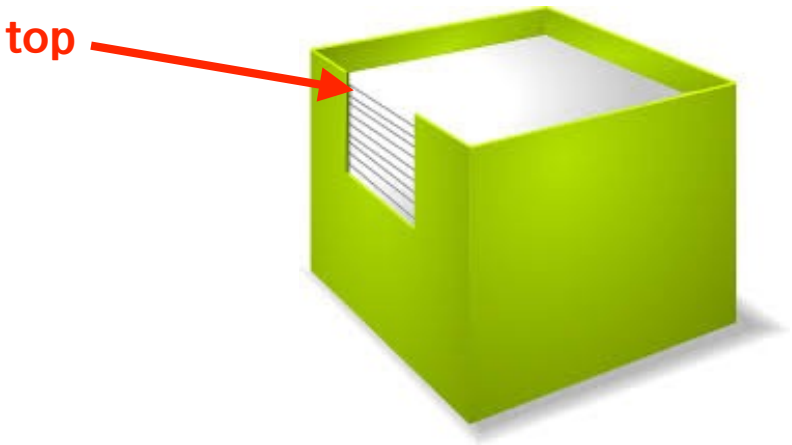
Introduction to Stack

Geun-Hyung Kim

UMCL @ Dong-Eui University

스택 정의 및 개념

- 스택 (Stack)
  - 데이터의 삽입과 삭제가 한쪽에서만 이루어지는 일종의 리스트
  - LIFO (Last-In, First-Out)
  - 데이터의 삽입/삭제가 일어나는 곳을 top 이라 함
  - 스택을 구현하는 대표적인 방법: 배열, 연결리스트

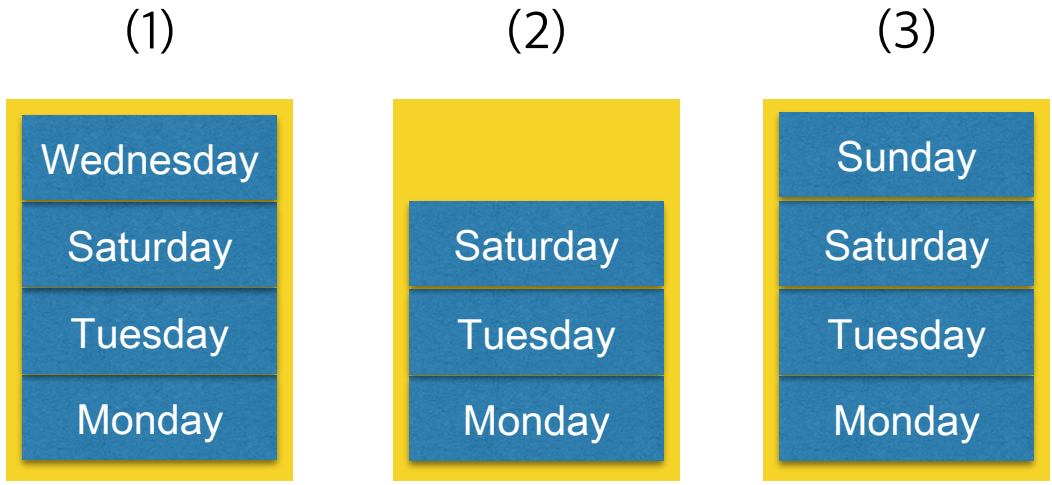


스택 연산

- push: 스택에 새로운 원소를 삽입하는 연산
- pop: 스택 top 위치에 있는 원소를 스택에서 제거하고 반환
- peek: 스택 top 위치에 있는 원소를 스택에서 제거하지 않고 반환
- isEmpty: 스택이 비어 있는지 검사하고 비었을 때 참 값을 반환
- isFull: 스택이 가득 찼는지를 검사하고 가득 차여 있을 때 참 값을 반환

push() and pop()

스택 연산 예




```
push("Monday");
push("Tuesday");
push("Saturday");
push("Wednesday");
```
(1)의 상태가 됨

```
char *str1 = peek();
```
(1)의 상태이고 str1 포인터에 "Wednesday" 저장

```
char *str2 = pop();
```
(2)의 결과이고 str2 포인터에 "Wednesday" 저장되고 Wednesday 값이 삭제

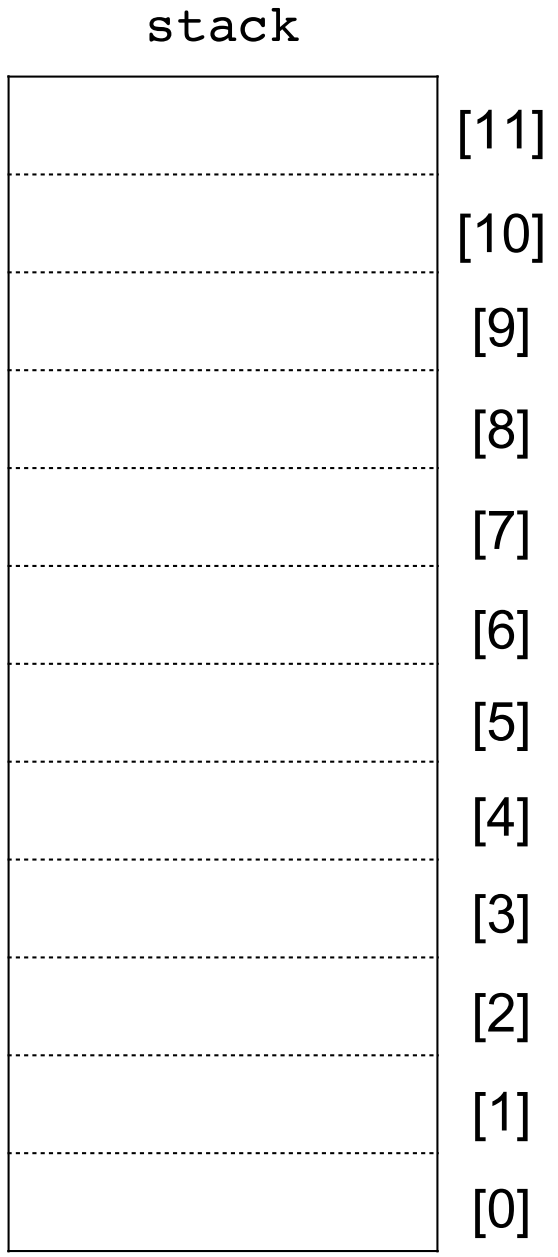
```
push("Sunday");
```
(3)의 상태가 됨

배열을 이용한 구현

스택 구현

```
#define MAX_STCK_SIZE 12

int stack[MAX_STACK_SIZE];

char stack[MAX_STACK_SIZE];

char *stack[MAX_STACK_SIZE];
```

```
#define MAX_STRING 100

struct newType{
    int student_no;
    char name[MAX_STRING];
    char address[MAX_STRING];
};

newType stack[MAX_STACK_SIZE];
```




```
int top = -1;
```

배열을 이용한 구현

stack 배열과 top이 전역변수로 선언된 경우

```
#define MAX_STACK_SIZE 100
int stack[MAX_STACK_SIZE];
int top = -1;

int isEmpty() {
 return top == -1;
}

int isFull() {
 return top == MAX_STACK_SIZE-1;
}

int push(int data) {
 if (isFull()) return -1;
 else {
  stack[++top] = data;
  return 1;
 }
}
```

else 내의 코드는
오른쪽 코드와 같은 의미

```
else {
   top++;
   stack[top] = data;
   return 1;
}
```

배열을 이용한 구현

stack 배열과 top이 전역변수로 선언된 경우

```
int pop() {
 if (isEmpty()) exit(-1);
 else {
   return stack[top--];
 }
}
```

else 내의 코드는 오른쪽 코드와 같은 의미

```
else {
   tmp = stack[top];
   top = top -1;
   return tmp;
}
```

```
int peek() {
 if (isEmpty()) exit(-1);
 else {
   return stack[top];
 }
}
```

연결리스트를 이용한 구현

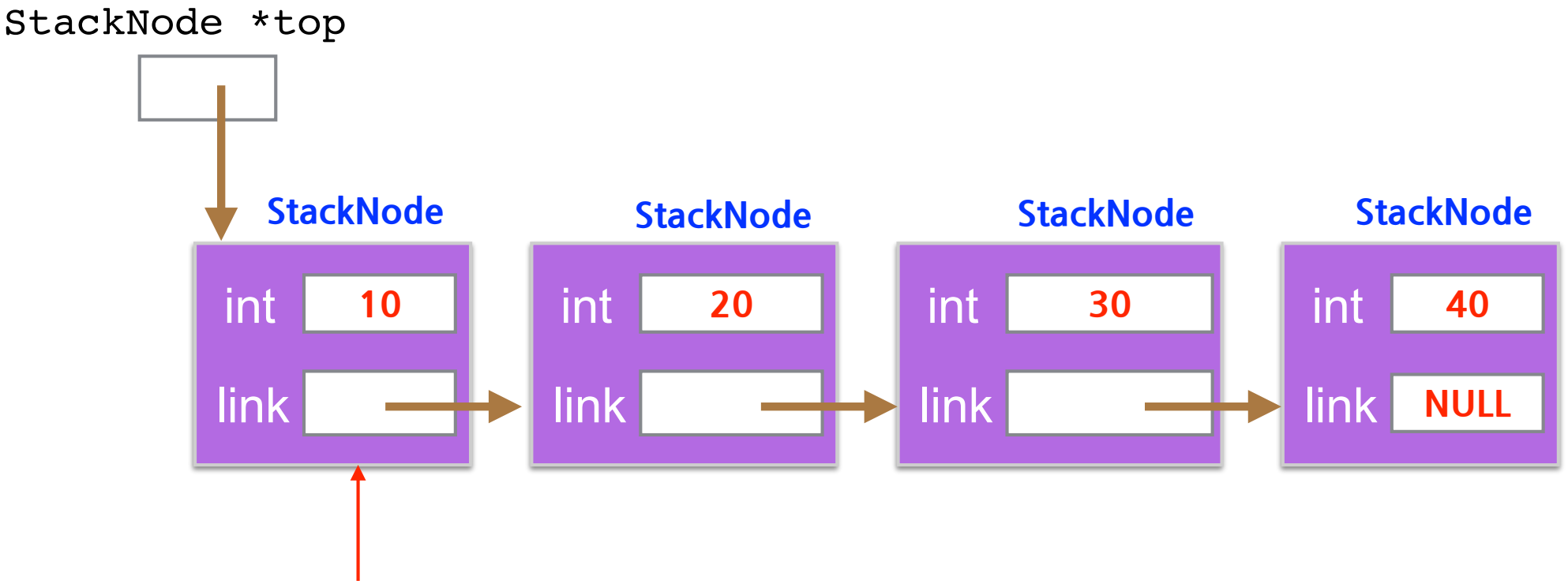
정수 데이터를 저장하는 스택

top이 전역변수로 선언된 경우

```
typedef struct stacknode {
   int data;
   struct stacknode *link;
} StackNode;

StackNode *top = NULL;      // top을 전역변수로 선언


int isEmpty() {
  return top == NULL;
}
```

정수 데이터

연결리스트를 이용한 구현



연결리스트의 맨 앞에 노드를 삽입하거나 삭제하기 용이하다.

그러므로 연결리스트의 맨 앞을 스택의 top으로 한다.

연결리스트를 이용한 구현

top이 전역변수로 선언된 경우

```
void push(int data) {
   StackNode *node = (StackNode *)malloc(sizeof(StackNode));
   node->data = data;
   node->link = top;
   top = node;
}
```

```
int pop() {
   if(isEmpty()) exit(-1);
   else {
      StackNode *node = top;
      int data = node->data;
      top = node->link;
      free(node);
      return data;
   }
}
```

```
int peek() {
   if(isEmpty()) exit(-1);
   else {
      return top->data;
   }
}
```

고민 거리…

1. 저장 장소를 전역변수의 형태로 선언하면 새로운 스택에 대해서 함수를 적용 가능한가?
2. 서로 다른 타입의 데이터를 스택에 저장하려고 할때 가능한가?

연결리스트를 이용한 구현

Stack 이 로컬 변수로 선언된 경우

```
typedef struct stacknode {
   int data;
   struct stacknode *link;
} StackNode;
```

```
typedef struct {
   StackNode *top;
} Stack;
```

```
void initStack(Stack *s) {
  s->top = NULL;
}
```

```
int isEmpty(Stack *s) {
  return s->top == NULL;
}
```

연결리스트를 이용한 구현


```
void push(Stack *s, int data) {
   StackNode *node = (StackNode *)malloc(sizeof(StackNode));
   node->data = data;
   node->link = s->top;
   s->top = node;
}
```

```
int pop(Stack *s) {
   if (isEmpty(s)) exit (-1);
   else {
      StackNode *node = s->top;
      int data = node->data;
      s->top = node->link;
      free(node);
      return data;
   }
}
```

```
int peek(Stack *s) {
   if (isEmpty(s)) exit (-1);
   else {
      return s->top->data;
   }
}
```

연결리스트를 이용한 구현

stack 배열과 top을 로컬 변수로 선언된 경우

push, pop, peek, is_empty, is_full 함수를 임의의 stack에서도 사용하기 위한 방법

```
#define MAX_STACK_SIZE 100
typedef struct {
   int stack[MAX_STACK_SIZE];
   int top;
} Stack;

void initStack(Stack*s) {
   s->top = -1;
}

int isEmpty(Stack *s) {
   return s->top == -1;
}

int isFull(Stack *s) {
   return s->top == MAX_STACK_SIZE-1;
}
```

연결리스트를 이용한 구현

stack 배열과 top을 로컬 변수로 선언된 경우

push, pop, peek, is_empty, is_full 함수를 임의의 stack에서도 사용하기 위한 방법

```
int push(Stack *s, int data) {
   if (isFull()) exit(-1);
   else {
     s->stack[++(s->top)] = data;
     return 1;
   }
}

int pop(Stack *s) {
   if (isEmpty()) exit(-1);
   else
     return s->stack[(s->top)—];
}

int peek(Stack *s) {
   if (isEmpty()) exit(-1);
   else
     return s->stack[s->top];
}
```

Summary of Stack

stack.h 파일 내용

```
#ifndef __STACK_H__
#define __STACK_H__

void initStack(Stack *s);
int  isEmpty(Stack *s);
int  isFull(Stack *s);
int  push(Stack *s, int item);
int  pop(Stack *s);
int  peek (Stack *s);

#endif
```

스택 응용
- 괄호 검사 -

개요

- 입력 수식의 괄호가 올바른지 검사
  - { A[(i+1)]=0 }
  - if((i==0) && (j==0)
  - A[(i+1]) = 0;

개요

- 입력 수식의 괄호가 올바른지 검사
  - { A[(i+1)]=0 }
  - if((i==0) && (j==0)
  - A[(i+1]) = 0;
- 단순하게 여는 괄호와 닫는 괄호의 개수 비교 만으로 부족
- 스택을 이용하여 검사
  - 여는 괄호는 스택에 push
  - 닫는 괄호가 나오면 스택에서 pop한 후 두 괄호가 같은 유형인지 확인
    - 유형이 다른 경우는 괄호를 잘못 사용한 것
  - 마지막에 스택에 남는 괄호가 있는지 확인
    - 마지막에 스택이 비어있지 않으면 괄호를 잘못 사용한 것

괄호 검사 프로그램


```
#include <iostream>
#include <string>
#include "stack.h"
#include "common.h"

char OPEN[] = "<({[";
char CLOSE[] = ">)}]";

bool checkMatching(char *str);
int checkOpen(char ch);
int checkClose(char ch);

int main()
{
  char *expr = GetStirng(); // commend 창에서 문자열 입력 받기
  if(checkMatching(expr))
    cout << "괄호 검사 성공" << endl;
  else
    cout << "괄호 검사 실패" << endl;
```

문자를 저장하는 스택에 대한 프로토타입이 정의되어 있음

checkOpen() / checkClose()


```
int checkOpen(char ch) {
 for(int i = 0; i < strlen(OPEN); i++)
  if (ch == OPEN[i]) return i; // <는 0,(는 1, {는 2, [는 3 반환
 return -1;
}
```

```
int checkClose(char ch) {
 for(int i = 0; i < strlen(CLOSE); i++)
  if (ch == CLOSE[i]) return i; // >는 0,)는 1, }는 2, ]는 3 반환
 return -1;
}
```

checkMatching()

스택 응용 예: 괄호 검사

```
bool checkMatching(char *str){
 Stack Stack;
 char open;
 int close;

 initStack(&Stack);
 for(int i = 0; i < strlen(str); i++) {
  if (checkOpen(str[i]) != -1)
   push(&Stack, str[i]);
  else if ( (close = checkClose(str[i])) != -1){
   if (isEmpty(&Stack)) return false;
   else {
     open = pop(&Stack);
     if (checkOpen(open) != close) return false;
   }
   return isEmpty(&Stack);
}
```

스택 선언

스택 초기화

문자가 여는 괄호인지 검사
여는 괄호이면 스택에 push

문자가 닫는 괄호인지 검사

닫는 괄호일 때 스택이 비워있는지 검사

닫는 괄호일 때 스택이 비워있지 않는 경우

스택 pop

스택 pop한 내용이
닫는 괄호인지 확인

모든 문자를 검사한 후 스택이 비워 있는지 확인

스택 응용
- 수식의 후위 표기 -

개 요

- 후위 표기법을 사용하는 이유
  - 컴파일러가 중위 표기법으로 작성한 수식을 어떤 순서로 계산되어야 알 수 있기 때문
  - 연산자의 우선 순위를 고려할 필요 없음

- 중위 표기법을 후위 표기법으로 어떻게 바꾸나?

- 후위 표기법으로 표현된 것을 어떻게 계산하나?

후위 표기 연산 예

2 3 4 * + : 2 + 3 * 4

a b * 5 + : a * b + 5

1 2 + 7 * : (1 + 2) * 7

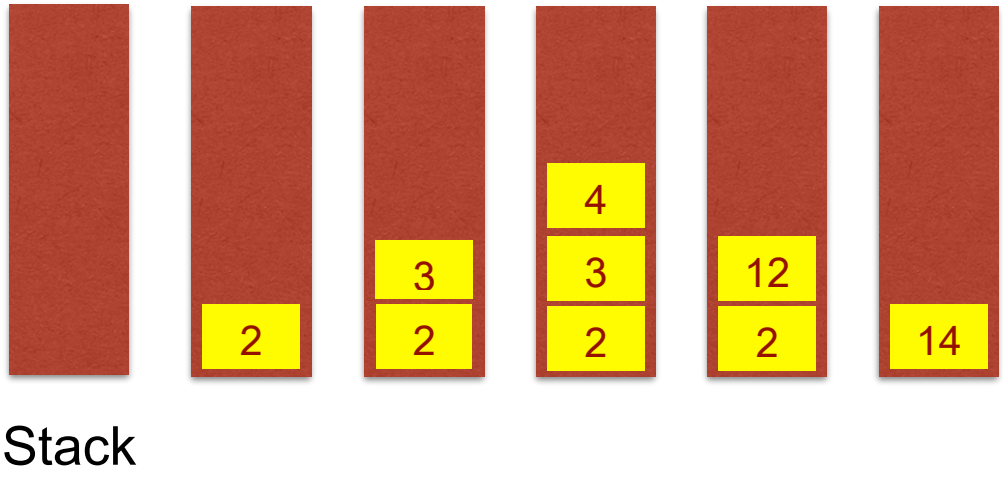
3 2 7 + * : 3 * (2 + 7)

3 2 7 * 5 / - : 3 - 2 * 7 / 5

8 2 / 3 - 3 2 * + : (8 / 2 - 3) + 3 * 2

- 알고리즘은 ?

2 3 4 * +



후위 표기식 연산하기

반환 자료 형: 정수
함수 이름: postfixCalc
입력 파러미터(1): 후위 표기로 작성한 수식 문자열

- **알고리즘을 구성하는 필요한 기능**
  - 수식 문자열에서 피 연산자와 연산자를 구분하는 기능 (피연산자와 연산자 모두 white space(“ “)로 구분된 문자열)
  - 피 연산자는 스택에 push
  - 연산자를 만나면 피 연산자 두개를 pop 하여 연산 후 연산 결과를 push

후위 표기식 연산 프로그램 코드

```
#include <iostream>
#include <stdlib.h>
#include “common.h”
#include “stack.h”

char OPERATORS[] = “+-*/“;

int checkOperator(char *ch) {
  for (int i = 0; i < strlen(OPERATORS); i++)
    if (ch[0] == OPERATORS[i]) return i;
  return -;
}

int main() {
  float result;
  cout << “후위 표기 수식을 입력하시요: “ << endl;
  char *expr = GetString();
  result = postfixCalc(expr);
  cout << “result of calculation: ” << result << endl;
  delete expr;
  return 0;
}
```

후위 표기식 연산 프로그램 코드

```
float postfixCalc(char *expr) {
 Stack stack;
 int op, i = 0;
 float result, loperand, roperand;
 char *str;

 init(&stack);
 str = strtok(expt," ");

 while (str != NULL) {
  if ( (op == checkOperator(str)) == -1) push (&stack, atof(str));
  else {
   roperand = pop(&Stack); loperand = pop(&stack);
   switch(op) {
    case 0: result = loperand + roperand; break;
    case 1: result = loperand - roperand; break;
    case 2: result = loperand * roperand; break;
    case 3: result = loperand / roperand; break;
   }
   push(&stack, result);
  }
  str = strtok(NULL," ");
 }
 return pop(&stack);
}
```

후위 표기식 변환: 괄호가 없는 경우

a+b : ab+

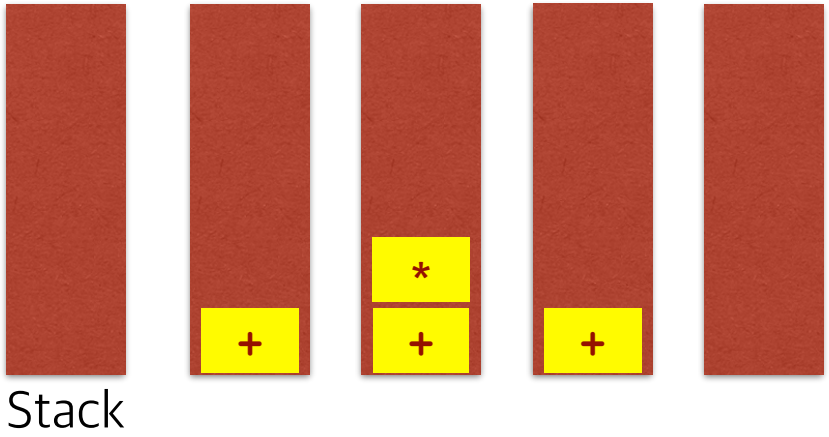
a+b*c : a+(b*c) -> a+bc* -> abc*+

a+b-c : (ab+)-c -> ab+c-

a*b-c : (a*b)-c -> (ab*)-c -> ab*c-

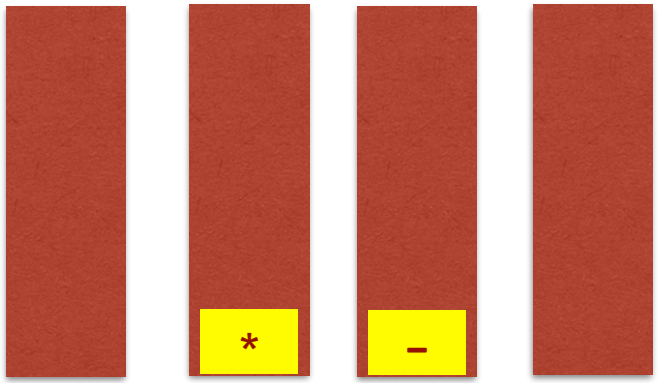
- 연산의 우선 순위 고려가 필요한가 ? 필요하다.
- 연산의 우선 순위 고려는 어떤 방식으로 ? 스택을 이용
- 알고리즘은 ?

a + b * c



abc*+

a * b - c



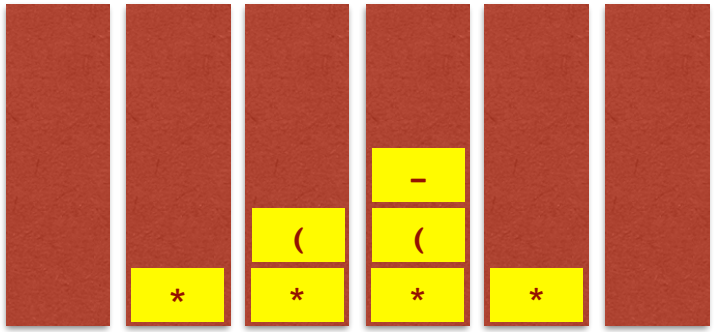
ab*c-

후위 표기식 변환: 괄호가 있는 경우

a*(b-c) : a*(bc-) -> abc-* (a+b)*c : (ab+)*c -> ab+c*

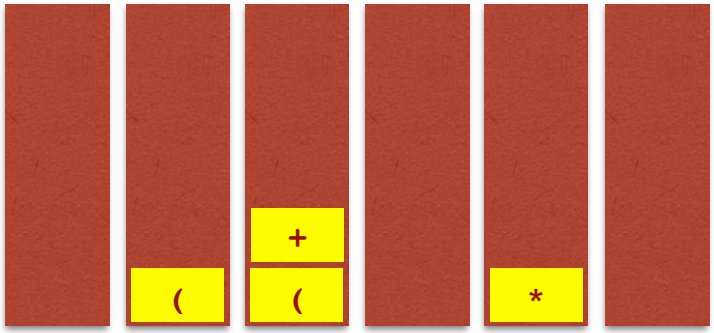
- 알고리즘은 ?

a * (b - c)



abc - *

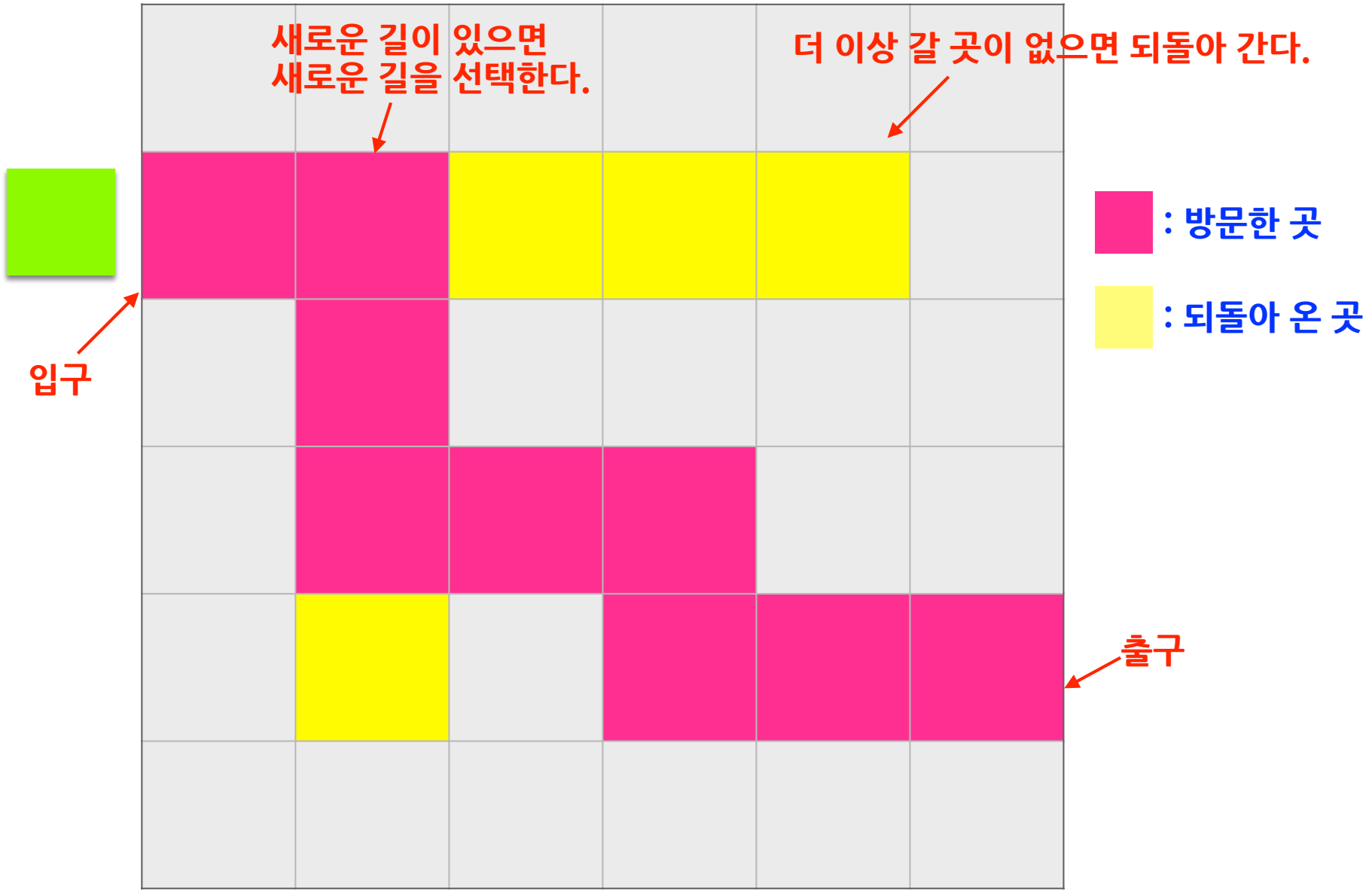
(a + b) * c



ab+c*

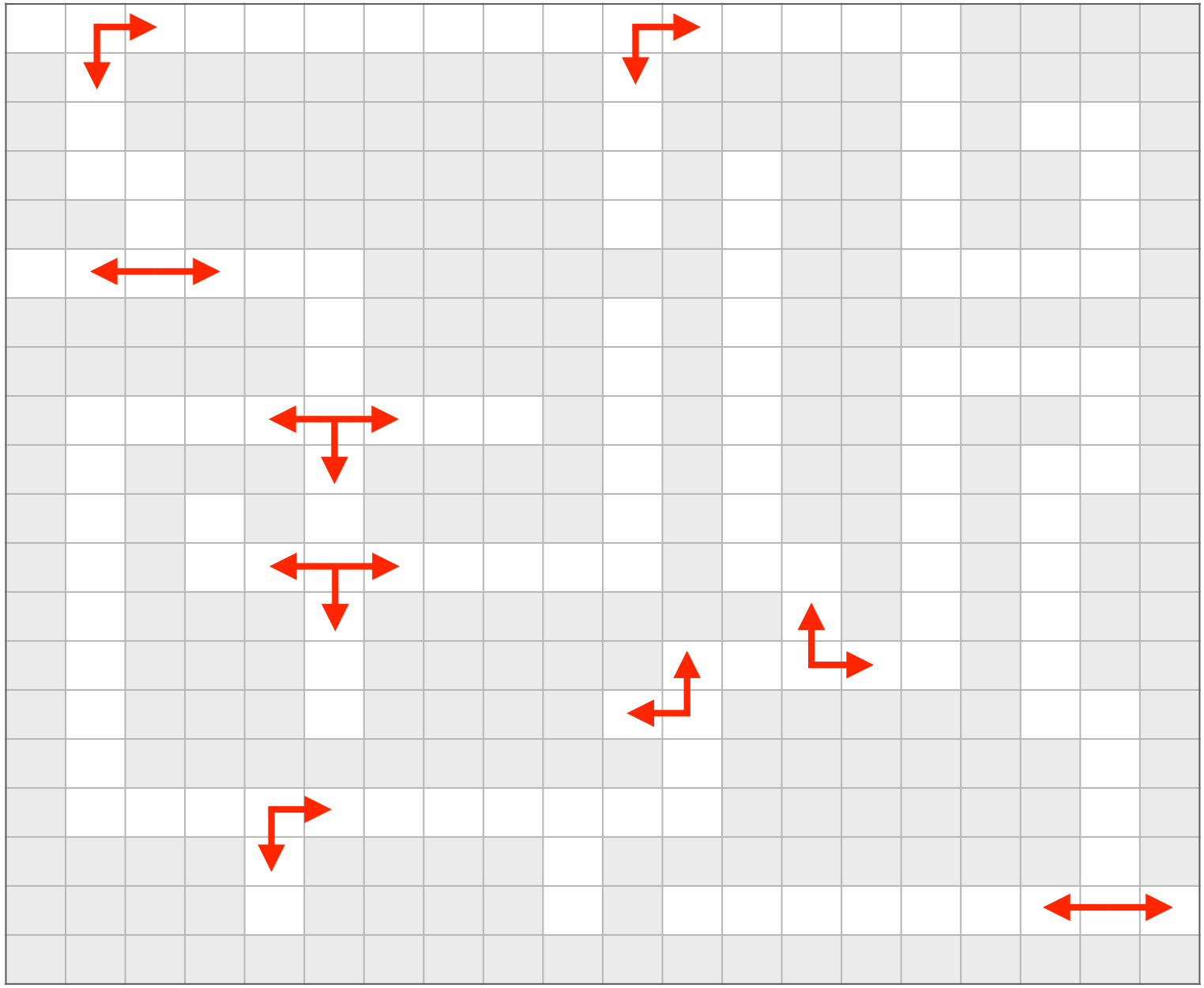
스택 응용
- 미로 찾기 -

개 요



개 요

스택응용: 미로 찾기



알고리즘 논의

스택응용: 미로 찾기

- 방문한 곳, 방문하지 않은 곳, 되돌아 나온 곳을 구분
- 매 위치에서 일정한 규칙(앞, 뒤, 좌, 우)으로 다음에 이동할 수 있는 방향을 검사
- 어떠한 방향으로도 갈 수 없으면 직전 위치로 되돌아 간다

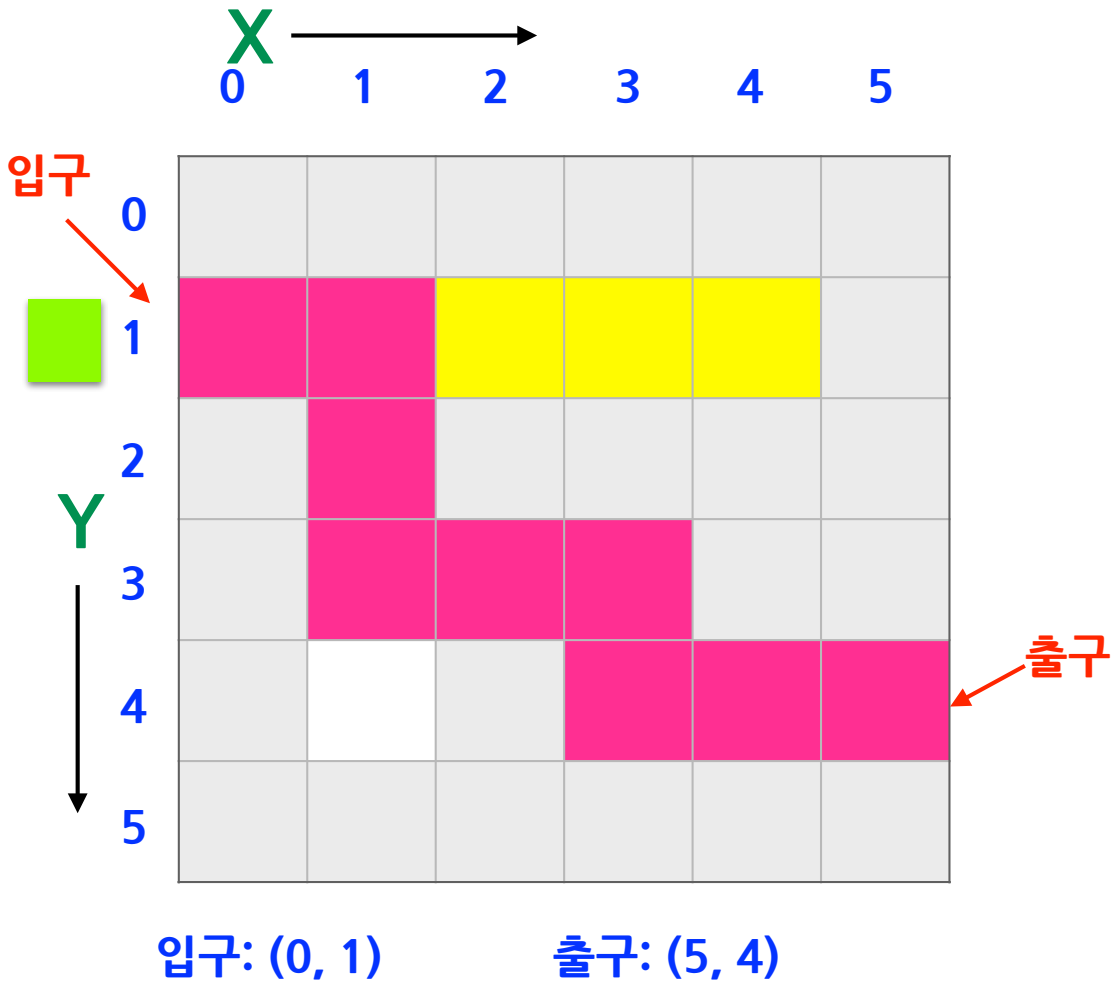
알고리즘

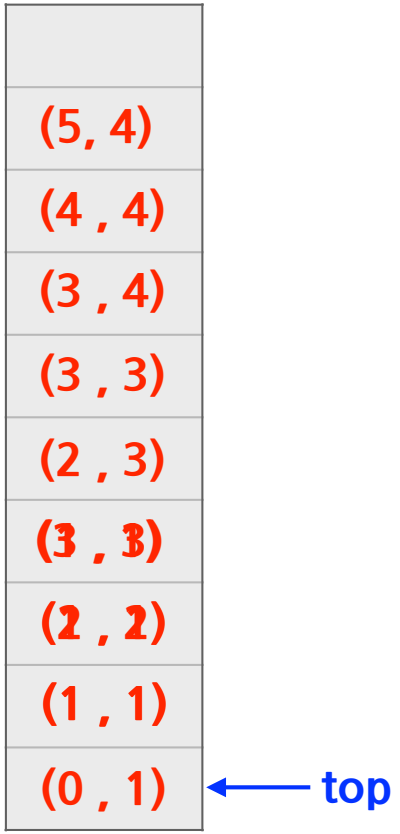
- **입구 위치로 이동**
- **다음을 반복**
  - **현재 위치를 방문했다고 표시**
  - **현재 위치가 출구가 아니면 다음을 수행하고 출구이면 종료**
  - **현재 위치에서 네 방향 (위, 오른쪽, 왼쪽, 아래)중 온 방향이 아닌 다른 방향으로 순서대로 다음을 수행**
    - 선택한 방향으로 이동이 가능한지 검사 후 가능하면, 현재 위치를 **stack에 push 후** 이동
    - 이동이 불가능한 경우: 벽, 되돌아 온 길, 미로 크기의 범위를 벗어 난 곳
  - **세 방향 중 어느 방향으로도 갈 수 없으면, stack에서 pop을 하여 바로 전 위치로 이동**

Stack 을 사용하는 경우: 되돌아가기 위한 위한 위치 정보를 저장하기 위해 사용

알고리즘과 스택

스택응용: 미로 찾기





Stack의 top은 항상 바로 전의 위치를 저장

프로그램을 위한 결정 사항

- 미로에 대한 정보를 컴퓨터 프로그램에서 표현하는 방법

  2차원 배열

- 방향 정보를 컴퓨터 프로그램에서 표현하는 방법

  네 방향을 나타내는 (0, 1), (1, 0), (0, -1), (-1, 0) 값으로 표현

- 길과 벽을 컴퓨터 프로그램에서 표현하는 방법

  배열에 저장 된 값으로 표현(길: 0, 벽 -1)

- 이미 방문한 곳과 되돌아 온 곳를 프로그램에서 표현하는 방법

  배열에 저장 된 값으로 표현 방문한 곳: 1 되돌아온 곳: 2

- 미로 내 위치를 표현하는 방법

  위치의 x 값과 y 값을 가진 구조체

프로그램 상세 코드

- 미로 내 위치를 표현하는 구조체

```
typedef struct position {
   int x;
   int y;
} Position;
```

- 방향 정보를 표현하는 방법 (0, 1), (1, 0), (0, -1), (-1, 0)

```
// down, right, up, left
int DIRECT[4][2] = {{0,1},{1,0},{0,-1},{-1,0}};
```

프로그램 상세 코드

- 다음 위치가 갈수 있는 곳 인지 확인하는 기능

```
bool movable (Position pos, int dir) {
  int nX = pos.x + DIRECT[dir][0];
  int nY = pox.y + DIRECT[dir][1];
  if (nX < 0 || nY < 0 || nX > MAZESIZE-1 || nY > MAZESIZE-1) {
     return false;
  }
  else if (maze[nY][nX] == 0) return true;
  else false;
}
```

- 입구 위치 지정할 변수 및 함수

```
Position in;  Position setEntry(int x, int y);
```

- 출구 위치 지정할 변수 및 함수

```
Position out;   Position setExit(int x, int y);
```

프로그램 상세 코드

- "maze.h" 파일 내용

```
#ifndef _MAZE_H_
#define _MAZE_H_

#define MAZESIZE 100            // 미로 크기
#define PATH  0                 // 아직 방문하지 않은 곳
#define VISITED 1               // 방문한 곳
#define BACKTRACKED 2           // 방문 후 되돌아 나온 곳
#define WALL -1                 // 방문할 수 없는 곳

typedef struct position {
   int x;
   int y;
} Position;

int DIRECT[4][2] = {{0,1},{1,0},{0,1},{-1,0}};
bool movable (Position pos, int dir);
Position nextPosition(Postion pos, int dir);
Position setPosition(int x, int y);

#endif
```

프로그램 상세 코드

```
int maze[MAZESIZE][MAZESIZE] = {
    {-1, -1, -1, -1, -1, -1},
    { 0,  0, -1,  0,  0, -1},
    {-1,  0,  0,  0, -1, -1},
    {-1,  0, -1,  0, -1, -1},
    {-1,  0, -1,  0,  0,  0},
    {-1, -1, -1, -1, -1, -1},
};
```

```
#include <stdio.h>
#include "stack.h"
#include "maze.h"

int main() {
 Stack s;
 initStack(&s);
 Position in = SetPosition(0, 1);
 Position out = SetPosition(5, 4);
 Position cur = in;
 bool moved = false;
 while(!isEqual(cur, in)) {
  maze[cur.y][cur.x] = VISITED;
  moved = false;
  for(int dir = 0; dir < 4; dir++) {
   if(movable(cur, dir)) {
    push(&s, cur);
    cur = nextPosition(cur, dir);
    moved = true;
    break;
   }
  }
```

프로그램 상세 코드

```
    if(!moved) {
      mazed[cur.y][cur.x] = BACKTRACKED;
      if(isEmpty(&s) {
        printf("No Path exists.\n");
        break;
      }
      cur = pop(&s);
    }
  }                 // end of while
}                   // end of main
```