**자료구조 02**

****

**중앙대학교**

**자료구조 05 분반**

**20165079 김영빈**

**20160125 김소령**

**목차**

과제

[**0. 과제 협업 방식** 3](#_Toc69052971)

[**1.** **계산기** 3](#_Toc69052972)

[**계산기 해결 방법** 4](#_Toc69052973)

[**2. 미로 탐색** 6](#_Toc69052974)

[**미로 탐색 해결방안** 6](#_Toc69052975)

[**3. 큐 운영하기** 8](#_Toc69052976)

[**큐 운영하기 해결방안** 8](#_Toc69052977)

[**4. 결과 화면** 10](#_Toc69052978)

[**1. 계산기** 10](#_Toc69052979)

[**2. 미로 탐색** 11](#_Toc69052980)

[**3. 큐 운영하기** 13](#_Toc69052981)

# **0. 과제 협업 방식**

이번 과제는 오프라인으로 만나서 자료의 저장방법이나 풀이 과정을 함께 고민하고 짝 프로그래밍을 수행하였습니다. 가령, 계산기의 경우 후위표현식으로 바꾸는 과정에서 어떤 점을 고려해야하는지 고민하고, 조건 별로 번갈아가면서 구현을 했습니다. 큐의 경우에도 큐의 저장방식, isFull일 때의 처리방식 등을 함께 고민한 뒤에 마찬가지로 조건별로 번갈아가면서 코딩을 했습니다. 미로 탐색은 가장 오랜 시간이 걸렸는데, 미로의 저장방식에 대해서 고민한 후, 구현은 각자 온라인으로 의견을 주고 받거나 코드 리뷰를 했습니다.

# **계산기**

계산식을 입력하면 결과를 돌려주는 계산기 프로그램을 작성하라.

– 사용되는 연산자는 +, -, \*, ^, %, ( ) 이다.

– 입력에 사용되는 수는 100 미만의 정수만 사용한다.

– ^ 는 제곱을 의미하며 3^5는 3 5이다.

^의 우선순위는 ( )보다 느리고 \* 보다 빠르다. 수식에서 ^는 최대 1회만 사용한다.

– %는 나머지를 구하는 연산으로, \*와 동일한 우선순위다.

– 수식에서 괄호는 최대 1회만 사용한다. 겹쳐진 괄호는 쓰지 않는다.

– 잘못된 수식이라면 어느 위치에 문제가 있는지 알려줘야 한다.

## **계산기 해결 방법**

Class Stack

- 연산자를 담을 스택을 생성한다. 강의 자료에서 제공된 코드를 참고.

Get\_postfix(equation)

- 주어진 식을 후위식으로 바꾸는 함수.

- 숫자를 만난 경우에는 무조건 식을 결과에 담아야 한다. 이 때, 숫자가 두 자리수일 때를 고려하여 temp라는 변수를 함께 운영하여, 연산자를 만난 경우 temp의 내용을 리스트에 추가한다.

- 연산자의 경우, 스택이 비어있거나(Stack.isEmpty() == True) 여는 괄호를 만난 경우에는 무조건 스택에 push한다.

- 그렇지 않은 경우, 스택의 가장 위의 연산자(Stack.peak())의 우선순위가 현재 만난 연산자보다 작을 경우에는 push한다.

- 닫는 괄호를 만난다면 여는 괄호가 나올 때까지 스택의 내용을 꺼내어(Stack.pop()) 리스트에 담는다.

- 비어있지 않고, 현재 연산자의 우선순위가 높거나 같다면 결과 리스트에 스택의 내용을 담는다.

- 마지막으로 남아있는 숫자를 결과 리스트에 담아주고, 스택의 내용도 차례로 비워준다.

Get\_value(equation)

- get\_postfix(equation)를 통해 후위식을 얻어낸 다음, 올바른 값을 얻어낸다.

- 반복문을 돌면서 연산자를 만나면 스택의 내용을 하나씩 꺼내어 차례로 연산하고 그 결과를 스택에 다시 담는다.

- 가장 마지막에 남은 하나의 숫자가 해당 후위식의 결과값이 된다.

Print\_error(space):

- 오류가 생긴 부분을 알려주는 함수이다. 틀린 부분의 위치를 알려주면 그 값만큼 공백을 출력하고 해당 위치에 오류가 있음을 알려준다.

Equation = 사용자로부터 연산을 입력받는다. 띄어쓰기 없이 입력받는다고 가정한다.

Errored = 해당 식이 error가 있는지 없는지를 저장하는 변수이다. 혹, 반복문의 마지막까지 error가 false라면, get\_value()를 통해 정확한 계산 값을 출력한다.

연산식의 마지막 요소가 +, -, \*, %, ^ 중 하나라면, 마지막 위치에 error를 표기한다.

Flag = [isnum, bracket opened] 각각 숫자인지 아닌지, 여는 괄호가 앞서 있었는지에 대한 Boolean 값을 저장해둔다.

반복문을 돌면서 숫자를 만나면 flag의 isnum은 true인 1로, flag의 bracket opened는 그 전의 값을 가져간다.

숫자가 아니면서 연산자 모음인 operator의 key 값 중 없는 수라면, 이 또한 해당 위치에서 오류가 있음을 출력하고 errored를 true로 바꾼다.

여는 괄호를 만났을 때, flag의 isnum이 true라면 오류임을 출력하고 errored를 true로 바꾼다. 곱셈이 생략된 것과 유사한 2(3+5)와같은 형태는 없는 것으로 한다. Flag의 bracket opened는 1증가시킨다. 0이 아닌 수는 모두 true이고, 여러 괄호가 중첩할 경우를 고려하기 때문이다.

닫는 괄호를 만났을 때 flag의 bracket opened가 false라면, 여는 괄호가 앞서 존재하지 않았다는 것이므로 오류를 표기한다. Flag의 bracket opened를 1감소시킨다.

이 외의 경우에 대해서는 flag의 isnum이 false라면 오류를 출력한다. 위의 조건들을 지나쳐온 현재의 요소는 +, -, \*, %, ^ 중 하나이기 때문에 숫자를 필요로하기 때문이다.

반복문을 돌 때마다 errored가 true라면 즉시 for문을 종료한다. 그렇지 않은 경우(통과한 경우)에는 space를 1 증가시킨다.

for문을 모두 돌고난 후에도 flag의 bracket opened가 true라면 무언가 닫히지 않은 괄호가 있다는 말이므로 오류를 표기한다.

# **2. 미로 탐색**

최적의 미로를 찾는 프로그램을 작성하라.

– 미로는 다음 페이지에 주어져 있다. 이 미로를 사용해야 한다.

– 미로의 길을 찾을 때에는 자신이 만든 스택을 이용해야 한다.

– 스택을 이용할 때마다 스택에서 어떤 정보를 PUSH하고 POP 하는지 화면에 표시해야 한다.

– 길은 여러 개가 있을 수 있다. 그 중 최종적으로 가장 빠른 길을 화면 에 표시해야 한다.

• 거리도 함께 표시한다.

• 미로 표시는 예쁘게 표시할 필요 없다. 알아볼 수 있는 수준으로만 표시 한다.

(+, -, | 활용)

## **미로 탐색 해결방안**

Class Stack

- 미로를 탐색할 때 사용하는 스택

Class Point

- 각 포인트들은 top, right, bottom, left의 값을 가진다.

- \_\_location은 포인트의 위치를 저장하고, icon은 지나온 길을 표시하기 위한 변수이다.

Class Maze

- 미로를 저장하기 위한 스택

- commands에는 주어진 미로를 포인트별로 입력한다. 각각은 위아래와 양옆의 값을 가지는데, 0번 인덱스에서의 -1은 위가 닫혔을 때, 1은 아래가 닫혔을 때, 2는 둘 다 닫혔을 때를 의미한다. 1번 인덱스에서 -1은 왼쪽이 닫혔을 때, 1은 오른쪽이 닫혔을 때, 2는 둘 다 닫혔을 때를 의미한다.

- composition은 9\*12의 0으로만 이루어진 이차원 리스트로 만들고, 각 포인트에 Point 클래스를 저장해둔다.

- set\_point를 통해 각 값들의 top, bottom, left, bottom 값을 세팅한다. (상기 -1, 1, 2의 값들로 구분)

- get\_path에서는 가장 먼저 start\_point를 저장한다. Path\_stack에는 [current, recent]의 꼴로push하는데, Current에는 현재의 위치를, recent에는 최근 방문한 위치를 저장한다.

- current가 end\_point에 도달할 때까지(path\_stack.isEmpty() == True) 반복문을 돌면서 스택을 운영한다.

- temp(=[top, bottom, left, right])는 경로 탐색을 위한 일시적인 리스트로 각 포인트들에서 길리 열려 있고, 최근 방문한 위치가 아니라면 해당 인덱스를 1로 바꾸어준다. 위아래 양 옆 중 여러 방향이 열린 경우도 있기 때문에 elif가 아닌 if로 연결한다.

- 만약 하나라도 열린 길이 있다면(sum(temp) == True, 즉 0이 아닐 때), 열린 방향들을 모두 path\_stack에 push한다. visited에 current를 추가하고, 이동한 곳의 위치가 end\_point와 같다면 final에 방문 경로를 추가한다. 스택의 내용을 pop하고, current와 recent를 재할당한다.

- 만약 모든 위치가 막혔다면(sum(temp) == 0 || not sum(temp)), 스택의 마지막 요소를 뺴내고, current와 recent를 재할당한다.

- print\_maze는 미로를 출력하는 함수이다. 각 포인트들을 확인하고, |, \_ 등을 사용하여 미로를 그린다. Icon 값을 확인하여, icon이 \*로 되어있는 경우에는 최종 경로에 해당하므로 이를 출력한다.

- print\_result는 결과를 출력하는 함수이다. Final\_path에 담긴 내용(end\_point까지 도달한 경우의 경로)의 길이를 출력하고, 거리는 final의 길이에서 1을 뺸다. 시작점은 거리에 해당하지 않기 때문이다. Final\_path가 비어있는 경우에는 경로가 없음을 출력한다.

# **3. 큐 운영하기**

1초에 1개의 문자가 입력되어 큐에 쌓이는 시스템에서 필요 한 수만큼 문자를 가져오는 프로그램

– 우리 시스템은 1초에 1개의 문자가 큐로 들어온다.

– 사용자가 필요한 데이터의 개수를 입력하면 큐에서 해당 데이터를 가 져온다.

– 입력 큐는 20 글자 크기를 갖는다. 입력 큐의 범위를 초과하면 받아들 일 수 없다(에러 표시).

– 원형 큐로 동작한다. 직접 원형 큐를 만들어야 한다.

시스템의 큐 입력

– 1초마다 A~Z 의 문자를 순차적으로 입력한다.

• 0초 : A를 큐에 넣는다. 1초 : B를 큐에 넣는다. 2초 : C를 큐에 넣는다…

사용자의 동작

– 숫자(1~9)를 넣으면 큐에서 해당 수만큼 가져와 화면에 표시한다.

– 0을 넣으면 큐에서 가져오지 않고 큐의 상태를 화면에 표시한다.

## **큐 운영하기 해결방안**

Class Queue

- 과제에서 사용할 원형 큐. 20개의 글자를 저장할 수 있는 큐를 만들되, 공간 하나는 비워두기 위하여 MAX\_QSIZE는 21로 한다.

- isEmpty는 font와 rear가 같을 때, isFull은 Front가 rear보다 1만큼 앞에 있을 때를 의미한다. 이 때 rear + 1를 MAX\_QSIZE로 나눈 나머지와 front를 비교한다.

- clear는 큐를 isEmpty의 상태로 만드는 함수이다.

- enqueue는 queue가 isFull이 아닌 상태에 대해서 rear를 하나 증가시키고 그 자리에 item을 위치시킨다.

- dequeue는 queue가 isEmpty가 아닌 상태에 대해서 fromt를 하나 증가시키고 그 자리의 item을 반환한다.

- peek은 queue가 isEmpty가 아닌 상태에 대해서 첫 요소를 반환한다.

- display는 queue의 내용을 보여주는 것인데, items 중 front+1부터 rear+1까지의 내용을 out 리스트에 담아 출력한다.

System = 운영될 큐의 이름

매번 Time.time()을 통해 start\_time을 기록하고, user\_input에 입력을 받는다.

입력 직후에는Interrupt\_time을 기록하여 그 차이만큼의 ALPHABET을 queue\_input\_index에 저장한다. 알파벳의 길이가 총 26임을 감안하여, Z이후에는 A부터 다시 기록될 수 있게 26으로 나눈 나머지를 저장한다.

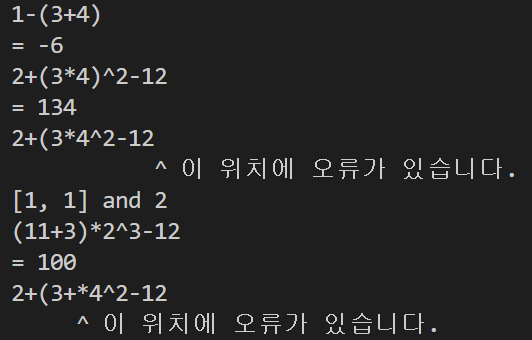
System이 가득차 있는 상황에서는 ADDQUEUE가 불가함을 알리고, 그렇지 않은 경우에 대해서는 앞선 queue\_input\_index만큼의 내용을 system에 enqueue한다. 그리고 queue\_input\_index를 1증가시킨다.

User\_input의 값만큼 system에서 dequeue하되, isEmpty의 경우에는 DELETEQUEUE가 불가함을 알린다.

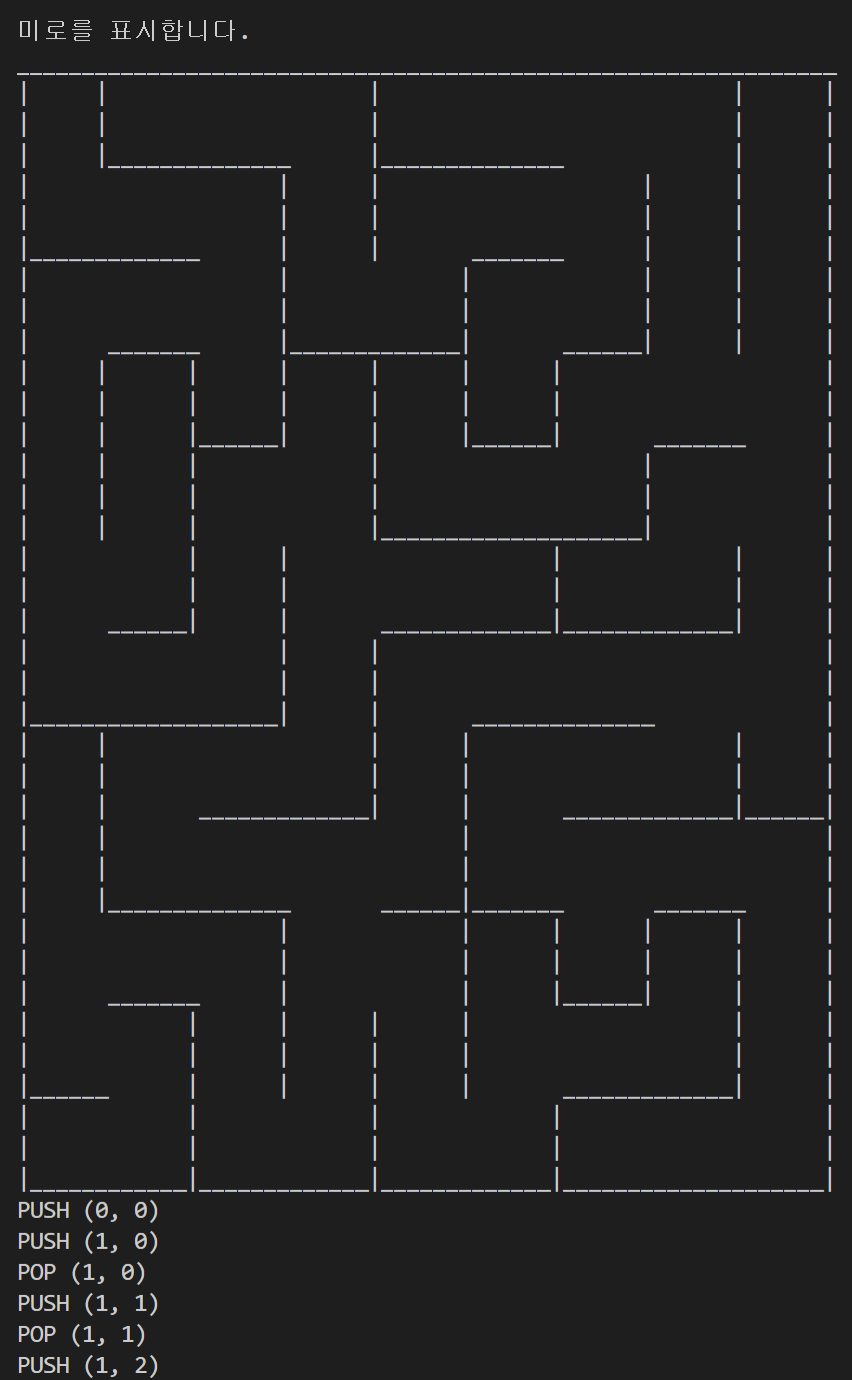
매 작업마다 system.display()를 통해 상황을 계속 보여준다.

# **4. 결과 화면**

## **1. 계산기**



## **2. 미로 탐색**





## **3. 큐 운영하기**

