알고리즘과 자료구조 워크북

교과목명 : 알고리즘과 자료구조
차시명: 11차시 큐와 스택 2

◈ 담당교수: 신일훈 **(서울과학기술대학교)**

◉ 세부목차
- 큐의 개념 및 연산
- 큐 구현
- 원형 양방향 연결리스트 구현

학습에 앞서

■ 학습개요

큐와 스택은 대표적인 선형 자료구조들이다. 스택은 아래쪽은 막혀 있고, 위쪽은 뚫려 있는 통 구조이며, 아이템의 추가와 제거가 모두 전면에서 발생하여 후입선출의 특성을 갖는다. 본 강에서는 스택의 개념 및 주요 연산을 이해하고 원형 연결리스트를 활용하여 스택을 구현한다. 또한 스택을 활용하는 대표적인 프로그램 예제 중 하나인 괄호 매치 검사 프로그램을 작성한다.

■ 학습목표

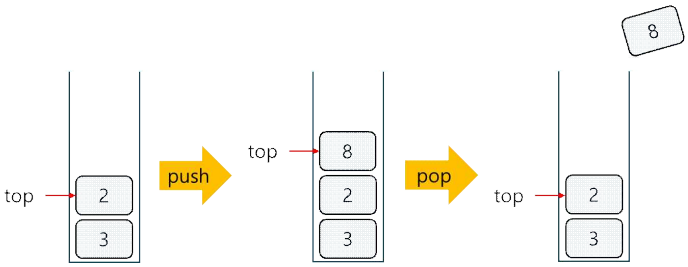
1	스택의 개념 및 연산을 이해한다.
2	스택을 구현할 수 있다.
3	스택 구현을 위한 원형 양방향 연결리스트를 구현할 수 있다.
4	스택을 활용한 괄호 매치 검사 프로그램을 구현할 수 있다.

■ 주요용어

용어	해설
스택	대표적인 선형 자료구조로서, 한쪽 방향만 뚫려 있는 통 구조이며, 아이템의 추가와 제거가 모두 전면에서 발생하여 후입선출의 특성을 갖는다.
후입선출(LIFO)	가장 나중에 들어온 아이템이 가장 먼저 나온다는 의미의 용어이다. 스택의 대표적인 특성이다.
push()	스택에 새로운 아이템을 추가하는 연산을 지칭한다,
pop()	스택에서 하나의 아이템을 꺼내는 연산을 지칭한다. 가장 나중에 추가된 아이템이 꺼내진다.

학습하기

1. 스택의 개념
   - 연결 리스트와 유사한 선형 자료구조
   - 한쪽만 뚫려 있는 통의 형태
   - 새로운 아이템의 추가는 스택의 전면으로만 가능
   - 아이템의 제거도 스택의 전면으로만 가능



   - LIFO(Last In First Out)의 특성
   - 추가, 제거, 검색, 크기 반환 등이 가능함
   - 함수호출정보를 저장하거나 그래프 탐색 (깊이우선탐색: death first search)을 수행할 때, 괄호 매치, 회문검사 등을 수행할 때 활용됨

2. 스택 구현

   가. 클래스 Stack 정의
   - 원형 양방향 연결리스트를 활용하여 구현.
   - push(): insert_front()로 구현
   - pop(): delete_front()로 구현
   - 멤버 변수
     - stack : 아이템들을 저장할 연결리스트 객체이며, 객체 생성 시에 빈 리스트로 초기화된다.
     - count : stack에 저장된 아이템의 개수
   - 메서드
     - 생성자
     - push()
     - pop()
     - get_size()
     - print()
     - search()

   나. 클래스 Stack 생성자 코드

```
import CList

class Stack :
        def __init__(self):
                self.stack  = CList.CList()
                self.count = 0
```

   - 원형 연결리스트 클래스가 CList.py에 구현되어 있다고 가정함.
   - CList.py에 필요한 메서드가 모두 구현되어 있다고 가정한다.
   - 스택 구현에 CList를 활용할 것이므로 해당 파일을 import 한다.
   - 스택이 생성될 때 비어 있는 상태이므로 stack을 빈 원형연결리스트로 생성하고 count를 0으로 초기

화한다.

다. 클래스 Stack의 push() 메서드 코드

- 스택의 전면에 삽입을 수행하는 메서드이다.
- insert_front() 메서드는 원형 연결리스트의 전면에 삽입하는 메서드이다.
- insert_front() 메서드를 호출하여 전면에 아이템을 삽입하고 count를 1 증가시킨다.

```
        def push(self, item):
                self.stack.insert_front(item)
                self.count += 1
```

라. 클래스 Stack의 pop() 메서드 코드

- 모든 노드들의 item 값들을 출력하는 메서드이다.

```
        def pop(self):
                if (self.count > 0):
                        self.count -= 1
                        item = self.stack.delete_front()
                        return item
                return None
```

- self.head가 0이면 None을 반환한다.
- self.head가 0보다 크면, count를 1 감소시킨다.
- delete_front()는 원형 연결리스트의 전면에서 아이템을 제거하여 반환하는 메서드이다.
- delete_front()를 호출하여 전면의 아이템을 제거하고 해당 아이템을 반환한다.

마. 클래스 Stack의 print() 메서드 코드

```
        def print(self):
                self.stack.print_list()
```

- 원형 연결리스트의 모든 노드들을 출력하는 print_list()를 호출하여 스택의 아이템들을 출력한다.

바. 클래스 Stack의 get_size() 메서드 코드

- 스택에 저장된 아이템의 개수를 반환하는 메서드이다.

```
        def get_size(self):
                return self.count
```

- self.count에는 아이템의 개수가 저장되어 있으므로 이를 반환한다.

사. 테스트 코드 및 실행 결과

```
if __name__ == '__main__':
        s = Stack()
        s.push('mango')
        s.push('apple')
        s.push('orange')
        s.print()
        for i in range(4):
                print(s.pop())

        s.push('apple')
        s.push('orange')
        s.print()
```

- 비어 있는 스택을 생성한 후, mango, apple, orange를 차례로 삽입한다.
- 스택을 출력하면 가장 처음에 추가한 mango가 제일 나중에 나와야 한다.
- for문을 반복하며 pop()을 4번 호출하면, orange, apple, mango가 차례로 제거되고 마지막에는 삭제가 실패한다.
- 이후 push() 메서드를 호출하여 apple, orange을 삽입하고 스택을 출력하면 orange, apple의 순서로 출력되어야 한다.
- 또한 이 테스트 코드는 __main__이 만족될 때 실행되므로, 이 파이썬 파일을 import할 때는 실행되지 않는다.
- 다음은 이에 대한 실행 결과이다.

```
orange  <=>  apple  <=>  mango
orange
apple
mango
None
orange  <=>  apple
```

3. 원형 양방향 연결리스트 구현
   - 스택의 구현에 필요한 insert_front() 함수만 정의한다. (다른 필요한 구현 내용은 9차시와 10차시를 참고)

   가. 클래스 CList의 insert_front() 메서드 코드
   - 전면 삽입을 수행하는 메서드이다.

```
    def insert_front(self, item):
        cnode = CNode(item, None, None)
        if (self.head == None):
            cnode.next = cnode
            cnode.prev = cnode
            self.head = cnode
        else:
            first = self.head
            last = first.prev
            cnode.next = first
            cnode.prev = last
            first.prev = cnode
            last.next = cnode
            self.head = cnode
```

- 추가할 item을 인자로 전달받고, CNode를 호출하여 item을 저장할 노드를 생성한다.
- 기존에 리스트가 비어 있는 경우에는 self.head를 생성한 노드로 설정하고 생성한 노드의 next, prev는 모두 자신을 가리키도록 하고 종료한다.
- 리스트가 비어있지 않은 경우에는 생성한 노드의 next가 기존의 첫째 노드(first)를 가리키도록 한다.
- 생성한 노드의 prev는 기존의 마지막 노드(last)를 가리키도록 한다.
- first의 prev는 생성한 노드를 가리키고, last의 next도 생성한 노드를 가리키도록 한다.
- 생성된 노드는 새로운 첫 노드이므로 self.head가 생성된 노드를 가리키도록 수정한다. (이를 제외한 나머지 코드는 insert_back()과 동일함.)

나. 테스트 코드 및 실행 결과

```
if __name__ == '__main__':
    c = CList()
    c.insert_front('mango')
    c.insert_front('orange')
    c.insert_front('apple')
    c.print_list()
    print(c.delete_front())
    c.print_list()
    c.insert_front('banana')
    c.print_list()
```

- 비어 있는 리스트를 생성한 후, mango, orange, apple을 차례로 전면에 삽입한다.
- 리스트를 출력하면 가장 먼저 추가한 mango가 제일 나중에 나와야 한다.
- delete_front()를 호출하면, 가장 나중에 추가한 apple이 제거된다.
- insert_front()를 호출하여 banana를 삽입하면 banana, orange, mango 순으로 출력되어야 한다.
- 다음은 이에 대한 실행 결과이다.

```
apple <=> orange <=> mango
apple
orange <=> mango
banana <=> orange <=> mango
```

4. 스택 활용 프로그램 – 괄호 매치 검사

가. 문제

- 스택을 활용하여 괄호 일치 여부 검사하기
- {{[()]}} : 일치
- {{[])) : 일치하지 않음

나. 아이디어

- 오른쪽 괄호를 만나면 직전 왼쪽 괄호와 매치가 되어야 함.
- 매치가 된 괄호들은 제거

  { { [( ) => { { [

- 이 과정을 반복하여 진행

- 주어진 괄호: {{[()]}}
- 매치 검사 과정

  { { [( ) => { { [=> { { [ ] => { { => { { } => { => { } => 매치됨

다. 알고리즘 (자연어로 표현)

1. 왼쪽 괄호 만나면 스택에 push
2. 오른쪽 괄호 만나면 스택에서 pop한 왼쪽 괄호와 매치하는지 비교
3. 매치하지 않으면 일치하지 않음.
4. 1 - 3 번 과정을 더 이상 괄호가 없을 때까지 반복
5. 만약 스택에 왼쪽 괄호가 남아 있으면 일치하지 않음
6. 스택에 왼쪽 괄호가 남아 있지 않으면 모든 괄호가 일치함.

라. 알고리즘 (의사코드로 표현)

```
1. def check_parenthesis (string) :
2.         stack = Stack()
3.         for char in string :
4.                 if (char is left_parenthesis) :
5.                         stack.push(char)
6.                 elif (char is right_parenthesis) :
7.                         left = stack.pop()
8.                         if (left is None or left not match char) :
9.                                 return False
10.        if (stack is not empty) :
11.                return False
12.        else :
13.                return True
```

- 3번 라인에서 string의 모든 문자에 대해 검사
- 4번 라인에서 만약 문자가 왼쪽 괄호에 속하면 스택에 저장
- 6번 라인에서 문자가 오른쪽 괄호에 속하면 스택에서 직전 왼쪽 괄호를 꺼내어 일치 여부를 검사함.
- 8번 라인에서 괄호가 서로 매치되지 않거나 꺼낼 왼쪽 괄호가 없으면 매치되지 않는다는 의미이므로 검사를 중단하고 False 반환
- 10번 라인 이후는 for문 완료 시에 실행됨. 즉 모든 문자의 검사가 완료되었음.
- 10번 라인에서 스택이 비어있지 않다면 왼쪽 괄호가 더 많다는 의미이므로 매치되지 않음
- 12번 라인에서 스택이 비어 있다면 왼쪽 괄호와 오른쪽 괄호가 모두 매치되었으므로 True 반환

마. 파이썬 구현 및 실행

```
1. import Stack
2.
3. def check_parenthesis(string) :
4.          s = Stack.Stack()
5.          parenthesis_dict = {
6.                  '{':'}' ,
7.                  '[':']',
8.                  '(':')'
9.          }
10.
11.          left_parenthesis = parenthesis_dict.keys()
12.          right_parenthesis = parenthesis_dict.values()
13.
14.          for char in string:
15.                  if char in left_parenthesis :
16.                          s.push(char)
17.                  elif char in right_parenthesis :
18.                          left = s.pop()
19.                          if (left == None     or      parenthesis_dict[left] != char) :
20.                                  return False
21.
22.          if s.get_size() > 0 :
23.                  return False
24.          else :
25.                  return True
```

- 1번 라인에서 스택을 사용할 것이므로 스택이 구현되어 있는 stack.py를 import.
- 4번 라인에서 스택을 생성
- 5-9번 라인에서 세 종류의 괄호에 대해, 파이썬 딕셔너리에 왼쪽 괄호를 key로, 오른쪽 괄호를 value로 저장.
- 11-12번 라인에서 keys(), values() 메서드를 호출하여 왼쪽 괄호들과 오른쪽 괄호들을 각각 list로 저장함.
- 14-20번 라인에서 string의 모든 문자에 대해 검사를 수행함.
- 15-16번 라인에서 만약 문자가 왼쪽 괄호에 속하면 스택에 저장
- 17-20번 라인에서 문자가 오른쪽 괄호에 속하면 스택에서 직전 왼쪽 괄호를 꺼내어 일치 여부를 검사함. 괄호가 서로 매치되지 않거나 꺼낼 왼쪽 괄호가 없으면 매치되지 않는다는 의미이므로 검사를 중단하고 False 반환
- 22번 라인 이후는 for문 완료 시에 실행됨. 즉 모든 문자의 검사가 완료되었음.
- 22번 라인에서 스택이 비어있지 않다면 왼쪽 괄호가 더 많다는 의미이므로 매치되지 않음
- 24번 라인에서 스택이 비어 있다면 왼쪽 괄호와 오른쪽 괄호가 모두 매치되었으므로 True 반환

- 다음은 테스트 코드와 실행 결과를 캡처한 것이다.
- 첫 번째만 일치하고 나머지는 일치하지 않는다.

```
1. if __name__ == '__main__':
2.         print(check_parenthesis('{a[c]b}'))
3.         print(check_parenthesis('{a[c]b}}'))
4.         print(check_parenthesis('{[a[c]b}'))
5.         print(check_parenthesis('{a[c]b)'))
```

```
True
False
False
False
```

연습문제

1. 선형 자료구조로서 LIFO의 특성을 갖는 자료구조는?.

정답 : 스택

해설 : 스택은 선형 자료구조로서 탑으로 데이터가 추가되고, 탑에서 데이터가 제거되므로 LIFO의 특성을 갖는다.

2. 원형 연결리스트로 스택을 구현한 경우, 스택에 새로운 아이템을 추가하는 push() 메서드의 최악의 시간복잡도를 구하시오.

정답 : O(1)

해설 : 아이템의 추가가 전면에서 발생하며, 원형 연결리스트를 사용하면 head 노드를 한번에 찾을 수 있다. 따라서 스택에 저장되어 있는 아이템의 개수와 관계없이 필요한 연산의 수가 일정하므로 시간복잡도는 O(1)이다.

3. 원형 연결리스트로 스택을 구현한 경우, push() 메서드의 알고리즘을 의사코드 형태로 표현하시오.

정답 :

```
def push(self) :
        self.queue.insert_front()
        self.count += 1          # 없어도 됨
```

해설 : 전면에 아이템을 추가하므로 insert_front() 메서드를 호출하면 된다. 만약 스택의 현재 아이템 개수를 관리하고 있다면, 해당 변수를 1 증가하면 된다.

정리하기

1. 스택은 선형 자료구조로서 전면으로 데이터가 추가되고 전면에서 데이터가 제거되므로 LIFO의 특성을 갖는다.
2. 스택을 구현할 수 있다.
3. 스택을 활용하여 괄호 매치 검사를 수행하는 프로그램을 작성할 수 있다.

참고자료

- 파이썬과 함께하는 자료구조의 이해, 양성봉, 2021, 생능출판

다음 차시 예고

- 해시 테이블에 대해 학습한다.