PULSE 2023 Fall

3주차 – 자료구조

본 강의는 자료구조를 구현 위주가 아닌 개념과 라이브러리 사용 방식에 대해 소개함.

목차

- C++: STL
  - ❖컨테이너 (Container)
  - ❖반복자 (Iterator)
  - ❖알고리즘 (Algorithm)
- Python – 표준 라이브러리
- 자료구조
  - ❖스택
  - ❖큐
  - ❖덱
  - ❖사전(Dictionary, Map)

C++: STL

C++: STL

- STL(Standard Template Library) – 표준 템플릿 라이브러리

- 일반적인 자료구조와 알고리즘을 구현한 라이브러리
- 크게 세 가지로 분류:
  - Containers
  - Iterators
  - Algorithms

Containers

- 자료를 저장하는 클래스 템플릿들의 집합
- 종류
  - Deque – 맨 앞, 맨 뒤에서 모두 O(1)로 삽입/삭제 가능.
  - Map – key-value로 구성. Key값은 유일해야 하나 value 값은 유일하지 않아도 됨.
  - Stack – LIFO 자료구조
  - Queue – FIFO 자료구조
  - 그 외에도 Pair, Vector, List, Set, …

Iterators

- Container 원소를 순회하는 방법을 추상화한 객체
- 종류
  - Forward Iterator - 한쪽 방향의 접근만 가능
  - Bidirectional Iterator – 양방향으로 접근 가능
  - Random-access Iterator – 임의 위치의 접근 가능

Algorithms

- 특정 활동을 수행하는 작업을 정의한 템플릿 함수
- 종류
  - Sort
  - Next_permutation
  - Unique
    - 같은 값이 연속적으로 있는 경우 제외함. Ex) {1, 2, 2, 3, 2} -> {1, 2, 3, 2}
    - 데이터 정렬 후 unique 사용 시 모든 원소가 유일해짐.
  - Fill – 원하는 범위의 값을 채움. 초기화에 유용
  - 그 외에도 lower_bound, upper_bound, reverse, …

Python

표준 라이브러리

- 파이썬 설치 시 자동으로 설치됨.
- Import 명령어를 이용하여 불러올 수 있음.
- Collections – Container datatypes
- 그 외에도 OS, shutil, glob, math, sys …

자료구조

스택

- LIFO(Last-in, First-out)
  - 가장 마지막에 넣은 원소가 가장 먼저 빠져나오는 구조
    - Push(x): 원소 x를 넣음
    - Pop: 가장 마지막에 넣은 원소를 빼냄

- 재귀적인 함수 호출에서 이러한 개념 사용됨.(함수 호출과 복귀 순서 관리)
- 활용 예시
  - 브라우저 방문기록
  - 후위 표기법 계산

스택 – C++

- Header: <stack>
- Class: stack<Type>
  - Int type 관리: std::stack<int>
- Member functions
  - push(x) : 원소 x를 stack에 집어 넣음
  - pop() : stack에서 가장 마지막에 집어 넣은 원소 제거
  - top() : stack에서 가장 마지막에 집어 넣은 원소 반환
  - empty() : stack이 비어 있는지 확인, 비어 있으면 true 반환
  - size() : stack의 현재 원소 개수 반환

스택 – Python

```python
my_stack = [] # 빈 스택 생성 (리스트 활용)

my_stack.append(1) # 스택에 원소 추가하기 (push)
my_stack.append(2)
my_stack.append(3)
print(my_stack)  # [1, 2, 3]

item1 = my_stack.pop() # 스택에서 원소 하나씩 빼내기 (pop)
item2 = my_stack.pop()
print(item1, item2)  # 3 2

is_empty = len(my_stack) == 0 # 스택이 비어있는지 확인하기
print(is_empty)  # False

stack_length = len(my_stack) # 스택의 길이 확인하기
print(stack_length)  # 1

if len(my_stack) > 0: # 스택의 맨 위 원소 확인하기 (peek)
    top_element = my_stack[-1]
    print(top_element)  # 1
else:
    print("스택이 비어 있습니다.")
```

큐

- FIFO(First-in, First-out)
  - 가장 먼저 넣은 원소가 가장 먼저 빠져나오는 구조
    - Push(x): 원소 x를 넣음
    - Pop: 가장 먼저 넣은 원소를 빼냄

- 데이터가 입력된 시간 순서대로 처리해야 할 필요가 있을 때 사용
- 활용 예시
  - 병원 대기 순번
  - 너비 우선 탐색 구현

큐 – C++

- Header: <queue>
- Class: queue<Type>
  - Int type 관리: std::queue<int>
- Member functions
  - push(x) : 원소 x를 queue에 집어 넣음
  - pop() : queue에서 가장 먼저 집어 넣은 원소 제거
  - front() : queue에서 가장 먼저 집어 넣은 원소 반환
  - empty() : queue가 비어 있는지 확인, 비어 있으면 true 반환
  - size() : queue의 현재 원소 개수 반환

큐 – Python

```python
import queue

my_queue = queue.Queue() # 큐 생성

my_queue.put(1) # 큐에 원소 추가하기
my_queue.put(2)
my_queue.put(3)

# 큐에서 원소 하나씩 가져오기
item1 = my_queue.get()
item2 = my_queue.get()
print(item1, item2)  # 1 2

queue_length = my_queue.qsize() # 큐의 길이 확인하기
print(queue_length)  # 1

is_empty = my_queue.empty() # 큐가 비어있는지 확인하기
print(is_empty)  # False
```

덱

- Deque – Double-ended queue
  - 양방향으로 넣고 뺄 수 있는 자료구조
  - 큐와 스택의 특성을 모두 가짐.
    - Push front(x): 원소 x를 앞에서 넣음
    - Pop front: 가장 앞에 있는 원소를 빼냄
    - Push back(x): 원소 x를 뒤에서 넣음
    - Pop back: 가장 뒤에 있는 원소를 빼냄

덱 – C++

- Header: <deque>
- Class: deque<Type>
  - Int type 관리: std::deque<int>
- Member functions
  - push_front(x), push_back(x) : 원소 x를 deque 앞/뒤에 집어 넣음
  - Pop_front(), pop_back() : deque에서 가장 앞/뒤에 있는 원소 제거
  - Front(), back() : deque에서 가장 앞/뒤에 있는 원소 반환
  - Empty(), size()

덱 – Python

```python
from collections import deque

my_deque = deque() # 덱 생성

my_deque.append(1) # 덱의 오른쪽 끝에 원소 추가하기
my_deque.append(2)
my_deque.append(3)
print(my_deque)  # deque([1, 2, 3])

my_deque.appendleft(0) # 덱의 왼쪽 끝에 원소 추가하기
print(my_deque)  # deque([0, 1, 2, 3])

my_deque.pop() # 덱의 오른쪽 끝에서 원소 제거하기
print(my_deque)  # deque([0, 1, 2])

my_deque.popleft() # 덱의 왼쪽 끝에서 원소 제거하기
print(my_deque)  # deque([1, 2])

length = len(my_deque) # 덱의 길이 확인하기
print(length)  # 2

first_element = my_deque[0] # 덱의 첫 번째 원소와 마지막 원소에 접근하기
last_element = my_deque[-1]
print(first_element, last_element)  # 1 2
```

사전 - Map & Dictionary

- 다양한 종류의 불변 객체를 사용해서 자료에 접근 가능
- Key – 사전에 사용되는 인덱스
- Value – Key가 가리키는 대상 객체
- Key-value pair – key와 value의 쌍

- 언어별 제공 방식
  - C++ – map container
  - Python – Dictionary 자료형

Map – C++

- Header: <map>
- Class: map<Type, Type>
  - Pair(String, int) type 관리: std::map<string, int>
- Member functions
  - insert({key, value}) : key가 중복되지 않은 경우 삽입
  - erase(key) : key 값을 기준으로 요소 삭제

Dictionary – Python

- 내장 자료구조인 dictionary 사용 시 key에 대한 value 가 없는 경우에 대한 처리가 추가적으로 이루어져야 함.(에러 발생)
- 파이썬의 내장 모듈인 collections의 defaultdict 클래스를 사용 시 위의 사항 고려 x

Dictionary – Python

```python
from collections import defaultdict, Counter

# defaultdict를 사용하여 기본값을 갖는 딕셔너리 생성
word_counts = defaultdict(int)
words = ["apple", "banana", "apple", "cherry", "banana", "apple"]

for word in words:
    word_counts[word] += 1

print(word_counts)  # defaultdict(<class 'int'>, {'apple': 3, 'banana': 2, 'cherry': 1})

# Counter를 사용하여 요소 개수 세기
word_counter = Counter(words)
print(word_counter)  # Counter({'apple': 3, 'banana': 2, 'cherry': 1})

# Counter를 사용하여 가장 많이 나오는 요소 찾기
most_common = word_counter.most_common(2)  # 상위 2개 요소 반환
print(most_common)  # [('apple', 3), ('banana', 2)]
```

참고 자료

- C++ STL reference
  - https://cplusplus.com/reference/
  - https://en.cppreference.com/w/

- 파이썬 표준 라이브러리 공식 문서
  - https://docs.python.org/3/library/index.html