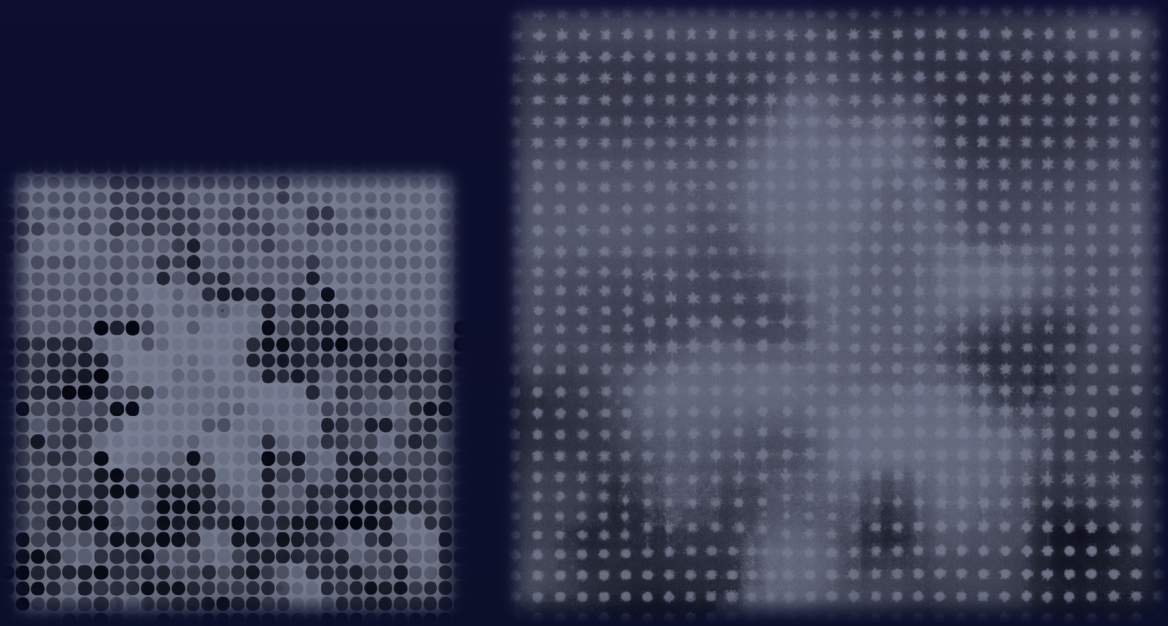
데이터 구조 실습 4주차

Data structure 1st Assignment introduce



2022 09 23

STL 소개

STL : Standard Template Libraty

다양한 종류의 C++컨테이너 클래스와 템플릿 알고리즘 제공

자주 활용되는 자료구조와 알고리즘을 제공

STL 소개

장점

임의의 데이터 타입을 갖는 자료구조를 만들 수 있음

소스크기의 축소

- 이미 STL에는 자주 사용되는 50여개 정도의 알고리즘과 다양한 데이터 구조들을 가지고 있음
- 이러한 STL에서 사용하는 자료구조와 알고리즘을 이용해서 소스 코드의 크기를 줄일수 있음

STL알고리즘

- 컨테이너들은 C/C++ 의 포인터와 배열에도 사용할 수 있으며, 변환 가능하므로 유연하게 사용
- Pre-optimized 알고리즘

STL 소개

STL구성요소

컨테이너 (Container)

- 데이터를 저장하는 객체들
- Pair, Vector, List, Stack 등

알고리즘 (Algorithm)

- 컨테이너의 데이터에 대한 정렬, 검색 알고리즘을 제공
- binary_search, find, merge 등

반복자

- 포인터와 유사하게 컨테이너 안의 원소를 순회하기 위한 객체
- 어떠한 컨테이너 타입에도 공통적인 인터페이스 제공

[STL] Pair

좌표와 같이 데이터를 (x, y) 쌍으로 표현해야 할 때 사용

특징 :

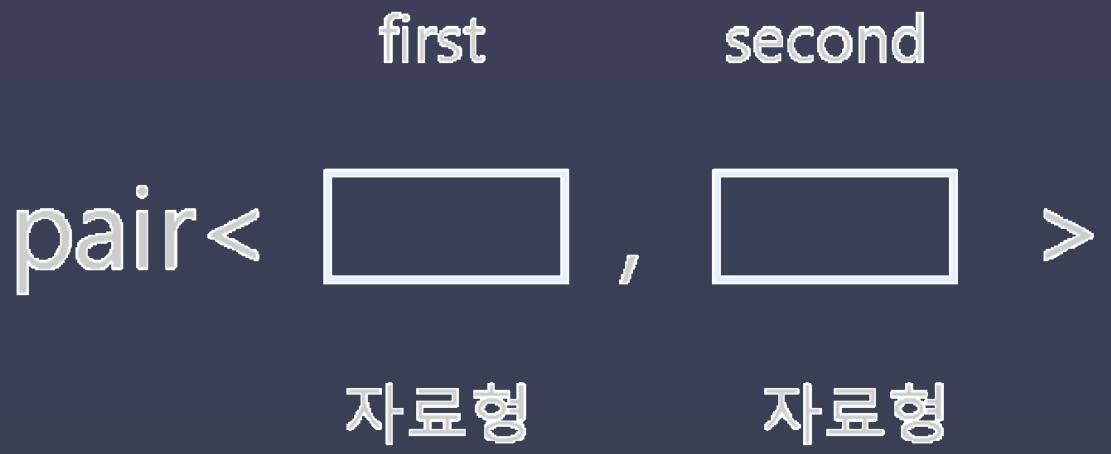
헤더 : #include<utility>

두 자료형을 묶을 수 있음

첫번째 자료형에 접근하기 위해서 'first', 두번째 자료형에 접근하기 위해서 second로 접근

make_pair(a,b)를 이용하거나 생성자를 통해 생성

pair안에 pair가 존재할 수는 없음



[STL] Stack

한 쪽 끝에서만 자료를 넣거나 뺄 수 있는 선형 형태의 Last In First Out 처리 구조

특징 :

헤더 : #include<utility>

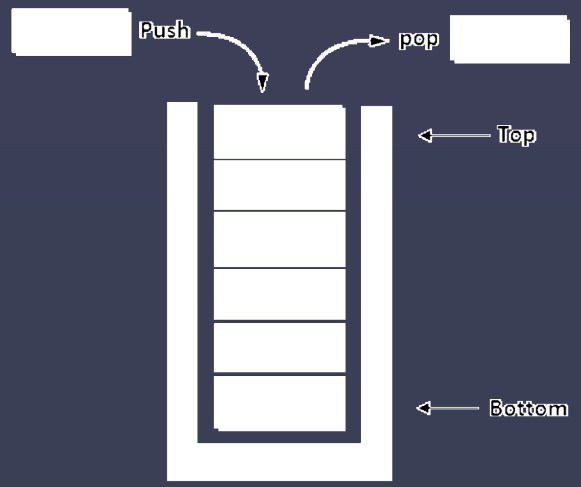
주요 멤버함수 :

push(a) : 스택에 데이터 a를 입력한다.

pop() : 스택의 Last In 데이터를 삭제한다.

top() : 스택의 Last In 데이터를 반환한다.

empty() : 스택이 비어 있는지 확인한다.



Example

```
#include<iostream>
#include<utility>
#include<stack>
using namespace std;

void main()
{
 stack<pair<int, char>> sp;

 for (int i = 0; i <4; i++)
    sp.push(make_pair(i, 'a' + i));

 while (!sp.empty())
 {
  cout << "Stack top <first,second>: <"
      << sp.top().first << ","
      << sp.top().second << ">" << endl;
  sp.pop();
 }
```

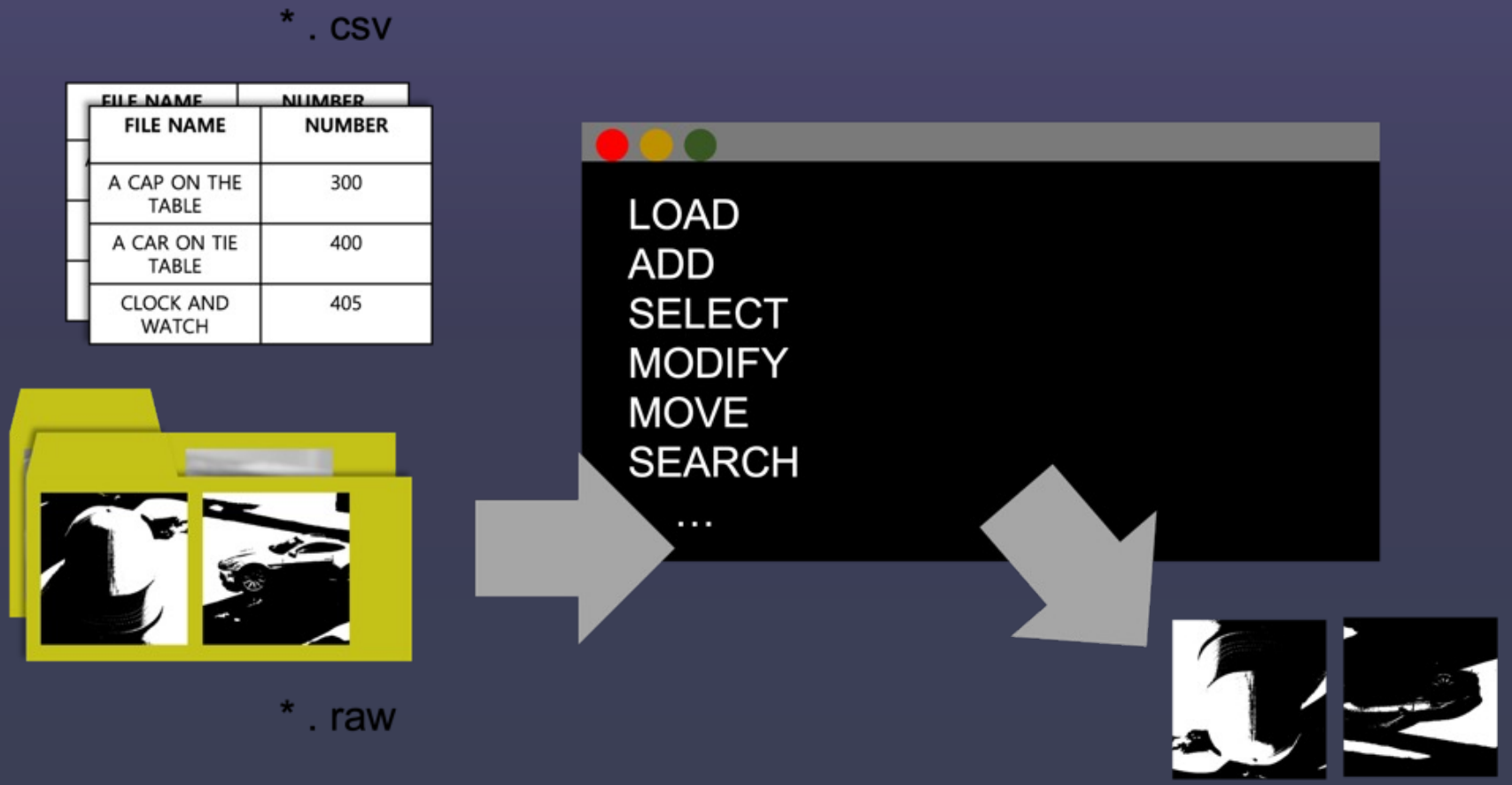
Console.

```
Stack top <first,second>: <3,d>
Stack top <first,second>: <2,c>
Stack top <first,second>: <1,b>
Stack top <first,second>: <0,a>
계속하려면 아무 키나 누르십시오 . . .
```

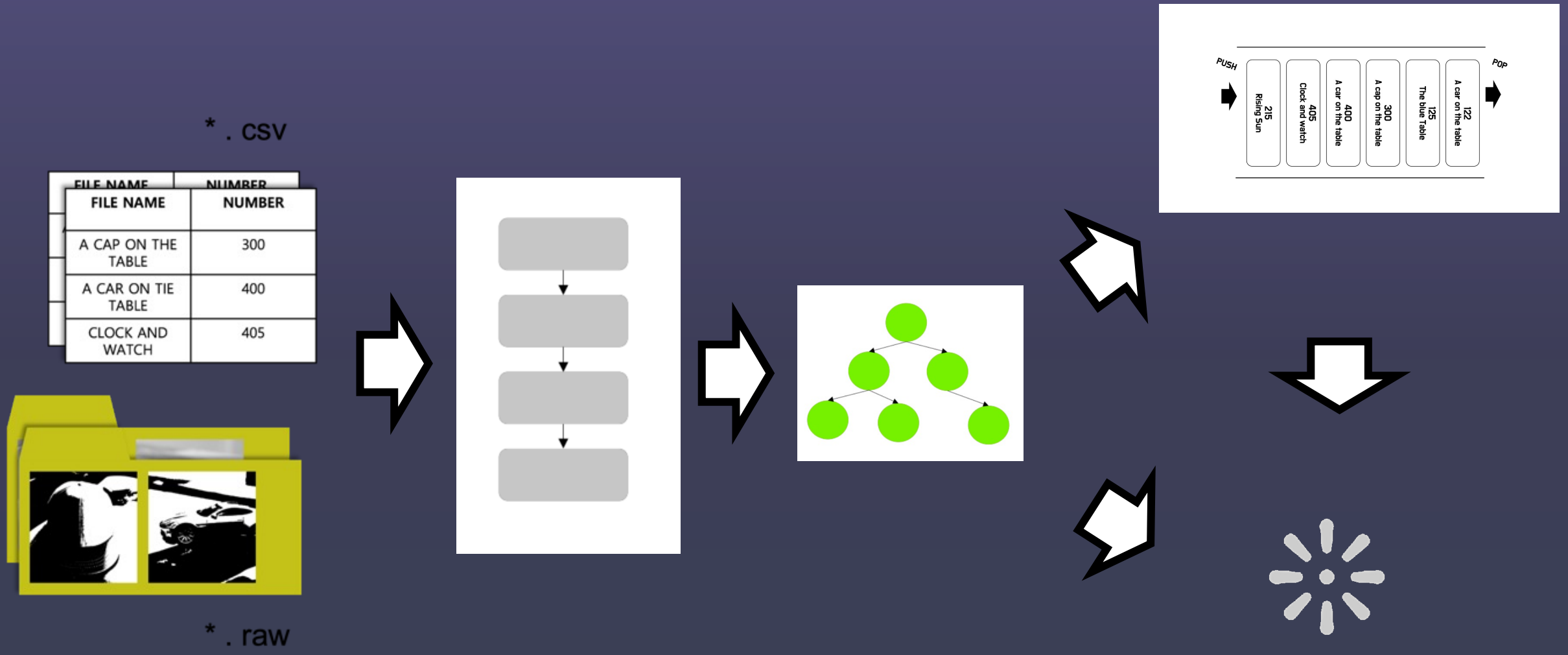
Project Details

Project 1

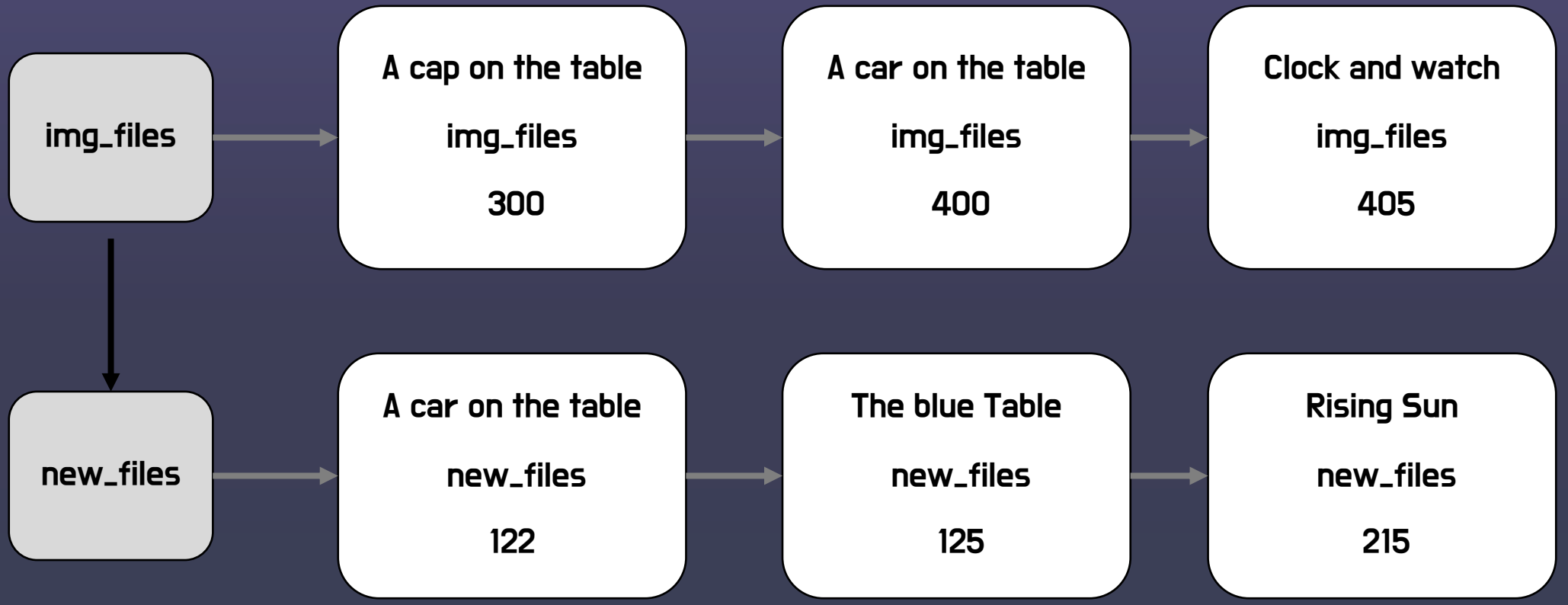
Project Details



Project Details



Linked List


img_files
A cap on the table
img_files
300
A car on the table
img_files
400
Clock and watch
img_files
405
new_files
A car on the table
new_files
122
The blue Table
new_files
125
Rising Sun
new_files
215

BST

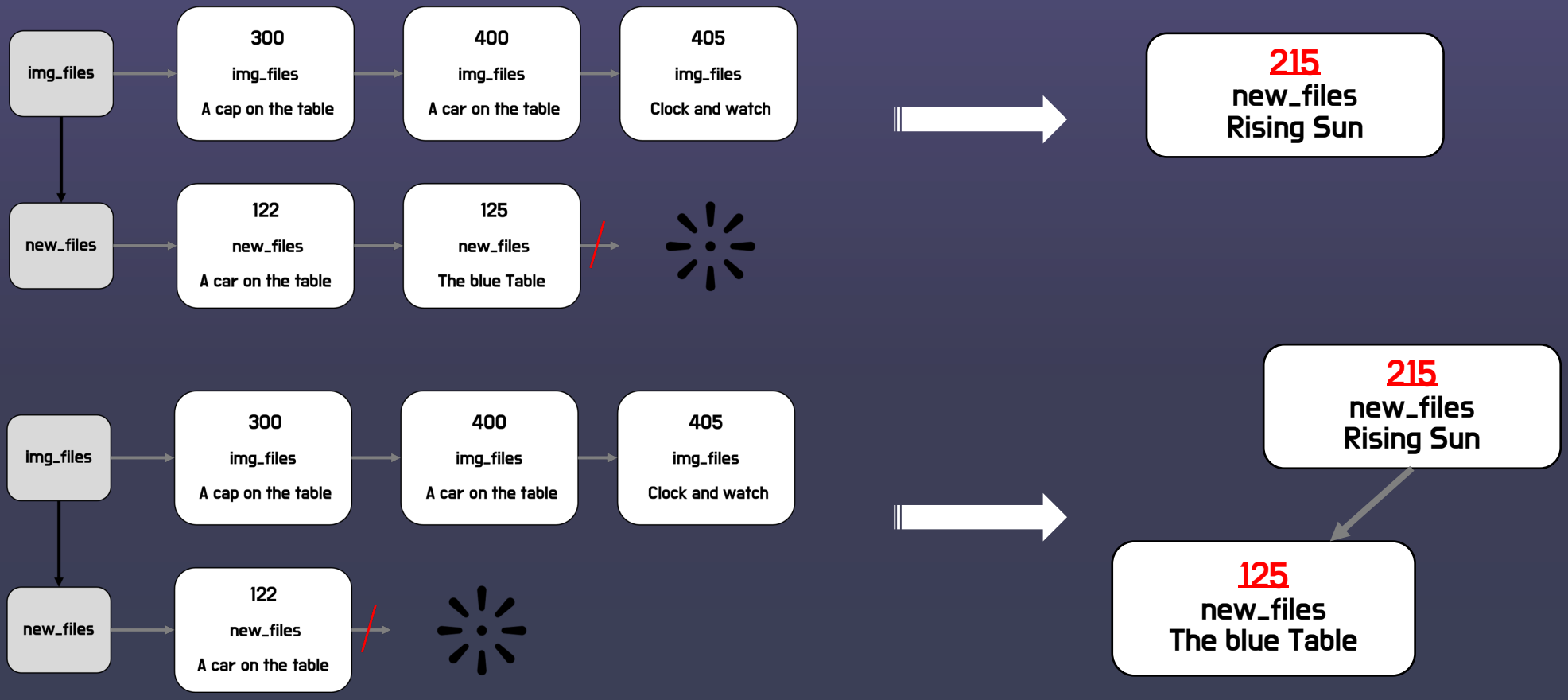
BST (Binary Search Tree)

Binary Tree의 한 종류

효율적인 탐색을 위해 일정한 규칙으로 구성

- 중복된 노드가 없음
- 노드의 왼쪽 서브 트리는 해당 노드의 값보다 작은 값들을 가진 노드로 구성
- 노드의 왼쪽 서브 트리는 해당 노드의 값보다 큰 값을 가진 노드로 구성
- 좌우 서브 트리는 다시 각각 BST를 구성

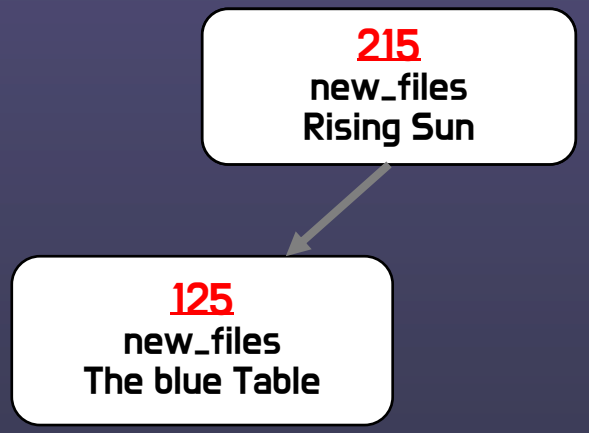
BST


img_files
300
img_files
A cap on the table
400
img_files
A car on the table
405
img_files
Clock and watch
new_files
122
new_files
A car on the table
125
new_files
The blue Table
215
new_files
Rising Sun
img_files
300
img_files
A cap on the table
400
img_files
A car on the table
405
img_files
Clock and watch
new_files
122
new_files
A car on the table
215
new_files
Rising Sun
125
new_files
The blue Table

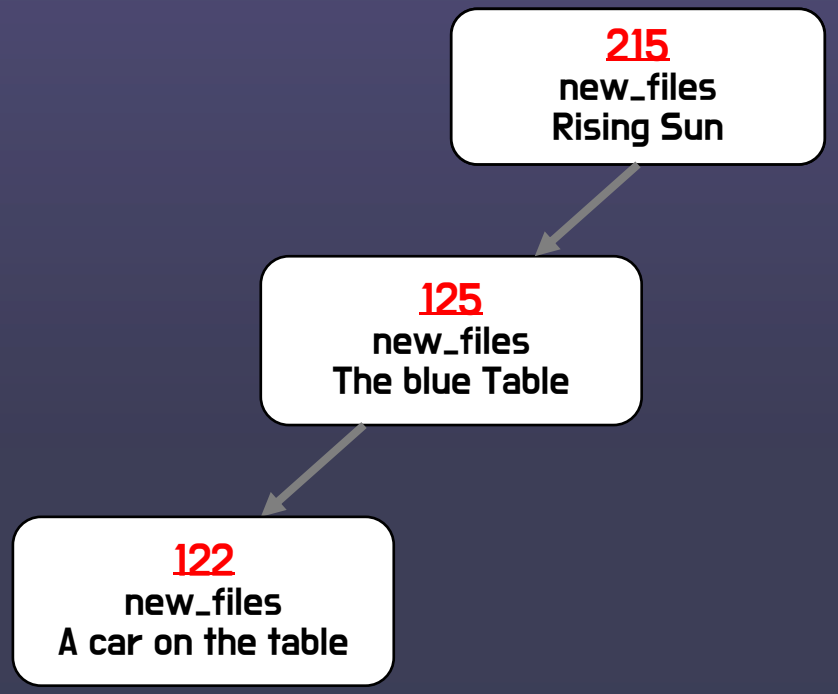
BST

215
new_files
Rising Sun

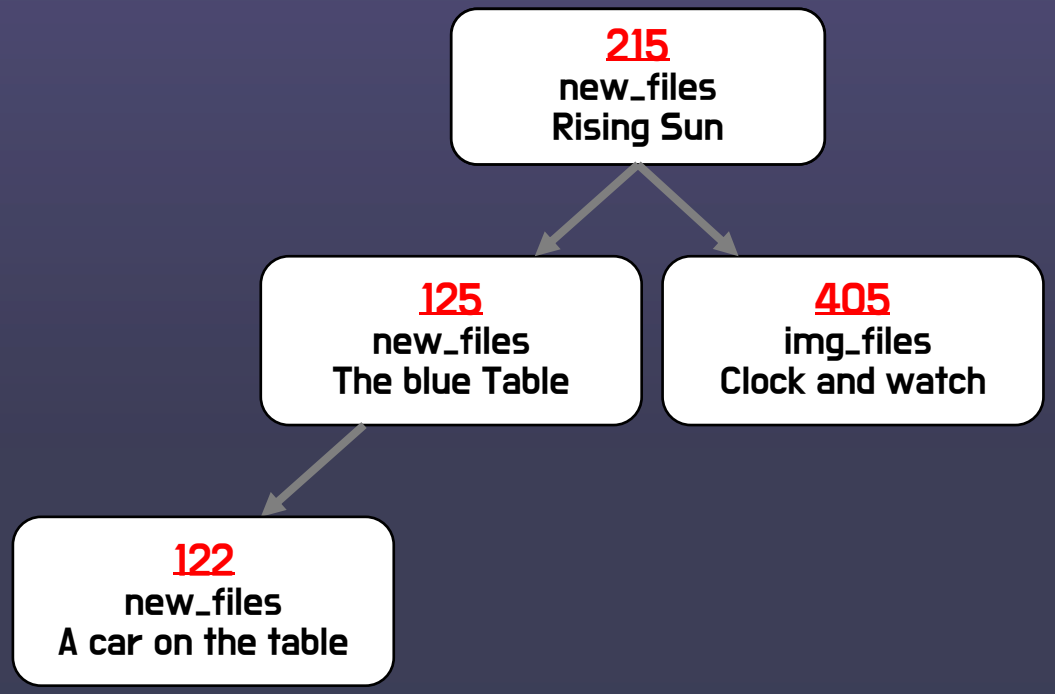
BST



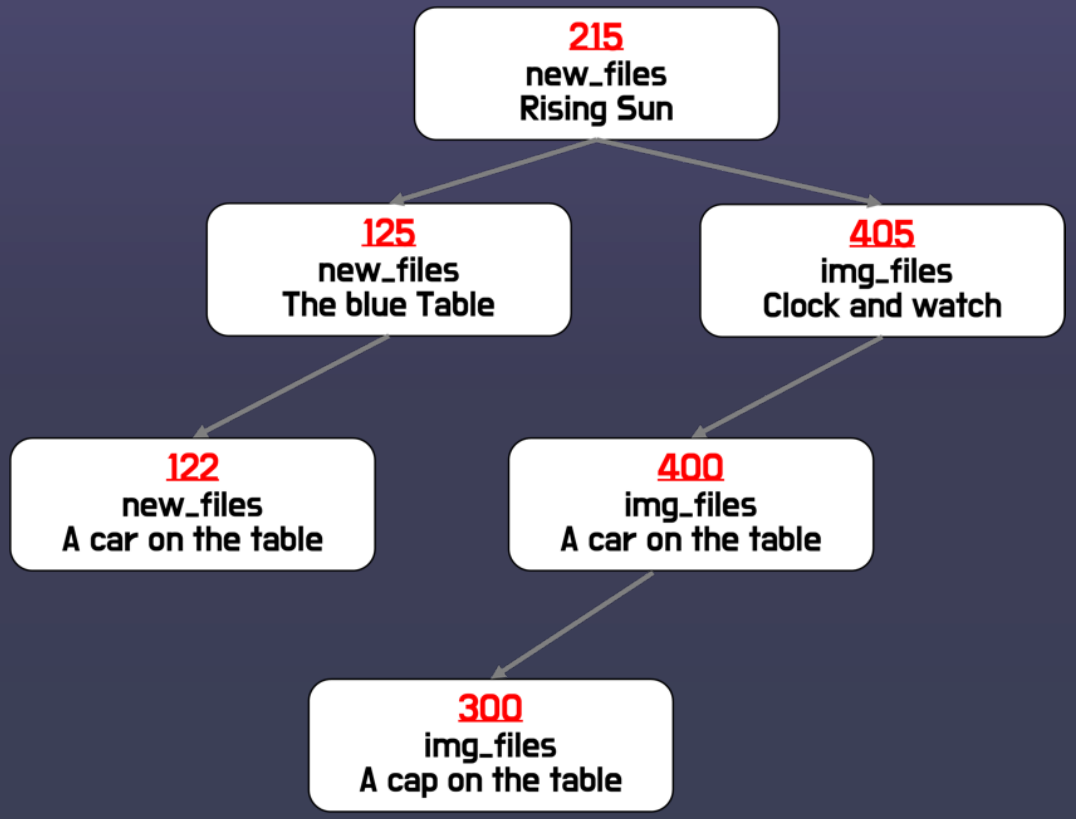
BST


215
new_files
Rising Sun
125
new_files
The blue Table
122
new_files
A car on the table

BST


215
new_files
Rising Sun
125
new_files
The blue Table
405
img_files
Clock and watch
122
new_files
A car on the table

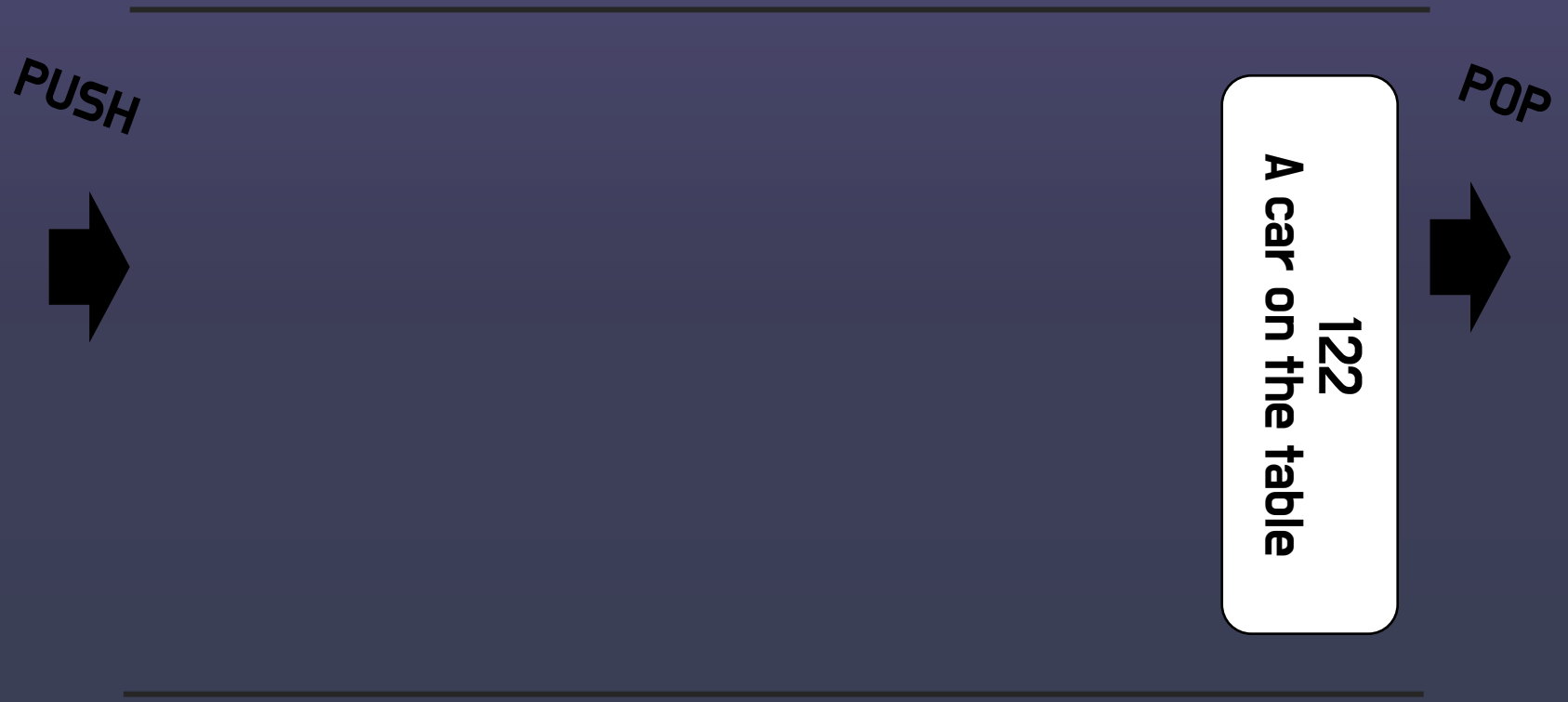
BST


215
new_files
Rising Sun
125
new_files
The blue Table
405
img_files
Clock and watch
122
new_files
A car on the table
400
img_files
A car on the table
300
img_files
A cap on the table

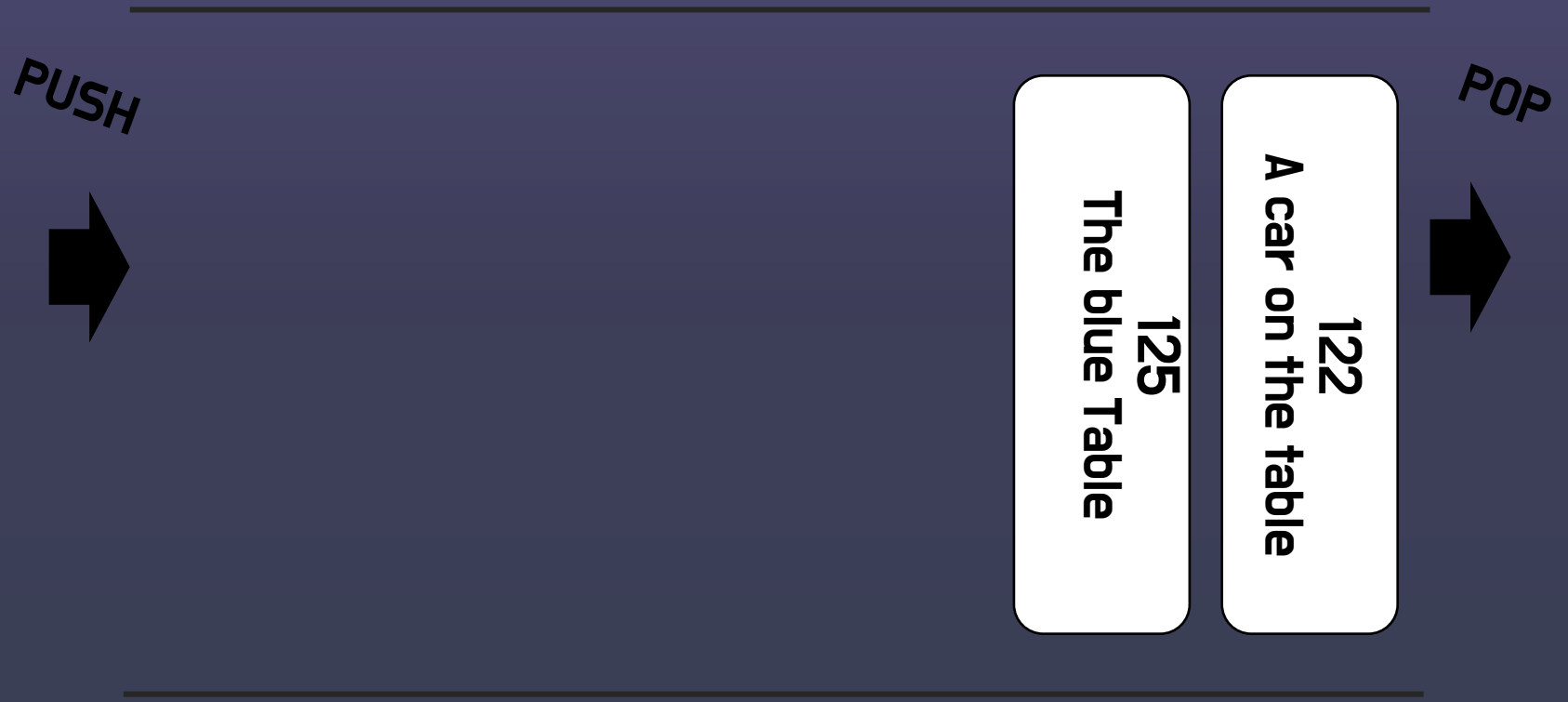
Queue


PUSH
215
Rising Sun
405
Clock and watch
400
A car on the table
300
A cap on the table
125
The blue Table
122
A car on the table
POP

Queue



Queue



Queue


PUSH
300
A cap on the table
125
The blue Table
122
A car on the table
POP

Queue


PUSH
400
A car on the table
300
A cap on the table
125
The blue Table
122
A car on the table
POP

Queue


PUSH
400
A car on the table
300
A cap on the table
125
The blue Table
122
A car on the table
POP

Queue


PUSH
215
Rising Sun
405
Clock and watch
400
A car on the table
300
A cap on the table
125
The blue Table
122
A car on the table
POP

Queue

215
Rising Sun

405
Clock and watch

400
A car on the table

300
A cap on the table

125
The blue Table

POP

122
A car on the table

Project Hands On

Project 1

Project Hands On

vector vs list

Vector

- 일반적인 배열처럼 개체들을 연속적인 메모리 공간에 저장
- 동적으로 확장/축소가 가능한 dynamic array
- 어떠한 순서로도 원소들을 순회할 수 있음
- 일반 배열처럼 인덱스 접근이 가능

List

- doubly linked list로 구현 됨
- 컨테이너의 어느 위치에서도 삽입/제거가 빠름
- 원소들의 컨테이너 내 순서 이동이 빠름, sort함수 사용 가능
- 노드 간의 연결에 관한 추가 메모리가 사용되어 vector보다 메모리 면 에서 손해
- 인덱스 접근이 불가능 하기 때문에 특정 원소 접근이 복잡

Project Hands On

txt 읽을 때 segmentation fault

ifstream 변수 선언 후 txt 파일을 eof를 이용하여 끝까지 읽을 때 segmentation fault 발생

보통 줄 단위로 읽어 strtok 등을 활용하지만 읽어온 줄이 공백이면 null 값을 읽어와 segmentation fault 발생

- 파일이 끝났으나 한번 더 읽으려고 할 때
- 파일에 공백줄이 존재할 때

읽어온 줄이 공백인 경우 continue 등으로 예외처리 필요

```
fin.open(command);
while (!fin.eof())
{
        fin.getline(cmd, 40);
        char* tmp = strtok(cmd, " ");
        if(tmp == NULL) continue;
```

Project Hands On

auto

선언한 변수나 람다식의 타입을 컴파일러에게 추론하도록 맡김

- const 및 &와 함께 사용가능

- Example)
  auto 변수명
  auto& 변수명
  const auto 변수명
  const auto& 변수명

```
int main() {
    vector<pair<int, char*> > v;
    int i = 10;
    char c = 'a';

    auto a1 = v.begin();  //vector<pair<int, char*> >::iterator a1
    auto a2 = i;          //int a2
    auto a3 = c;          // char a3
}
```