REPORT

**자료구조Assignment 02**

Data Structure Assignment 02Code Description



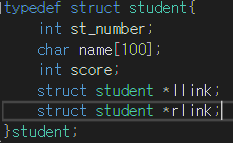
|  |  |
| --- | --- |
| 15조 | |
| 2015410056 | 김지윤 |
| 2015160022 | 박민정 |
| 2017320178 | 조재우 |
| 2014130341 | 한동희 |
| 제출 일자 | 2018/05/09 |
| 담당 교수 | 서영덕 교수님 |

개발 언어: C

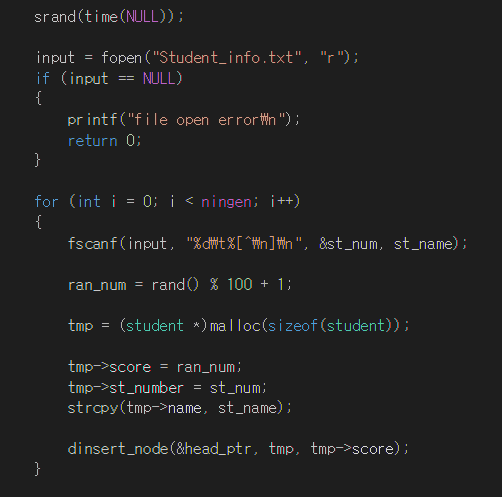
개발 환경: Visual Studio 2017 (Prob#03), gcc (Prob#02)

**[Prob#01]:** 이중 연결 리스트를 활용하여 학생 성적 리스트 만들기

1. 구조체 설정

 학생의 정보를 받고 리스트 구현을 위한 student 구조체를 만든다. 구조체의 요소로는 과제#1 과 같이 txt파일로부터 읽어 들이기 위해 학생의 학번, 이름을 위한 구조체와 랜덤함수를 통해 main에서 받을 int형 점수 구조체, 그리고 double linked list 구현을 위해 구조체의 주소인 student \* 형 변수로 llink, rlink를 구현한다.

1. 리스트 구현
2. Main 함수: 파일 읽기와 노드 생성

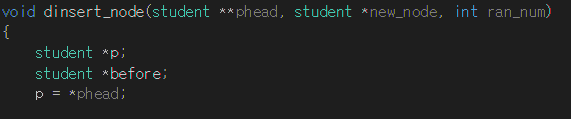


먼저, txt파일을 읽은 후에 오류검사를 한 후 위에서 선언 해 주었던 구조체에 값을 넣어 준다. 파일을 읽을 때마다 새로운 student구조체를 선언 해 준 뒤, 교수님께서 주신 형식으로 파일을 읽은 뒤 데이터(학번, 이름)를 해당 구조체에 저장한다. 동시에 랜덤으로 score를 생성 해 배정을 해 준 뒤 리스트에 바로 넣어 준다(dinsert\_node 함수가 해당 함수).

위의 경우 ningen 이라는 값을 학생 수 값으로 설정해 주어 text파일에 있는 학생 수를 지정했지만, 학생 수를 모르는 경우 while문을 사용하여 fscanf의 값이 NULL이 되는 경우까지 탐색을 하면 된다.

1. Void dinsert\_node(student \*\*phead, student \*new\_node, int ran\_num): head 생성과, 리스트 탐색 후 적절한 위치에 노드 배정

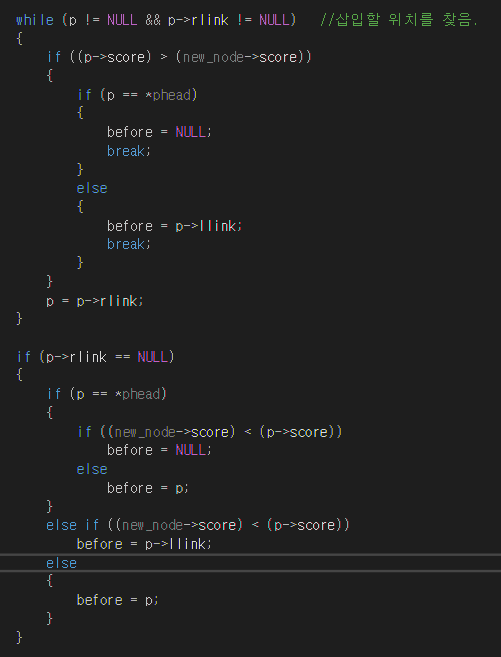
먼저, dinsert\_node 함수의 parameter는 다음과 같다.



첫 위치부터 탐색을 하여 해당 노드를 넣을 위치를 찾기 위해 phead값을 넣어주는 데, 이 때의 parameter는 이중포인터가 된다. 또한 새로 리스트에 삽입하기 위한 new\_node와 해당 위치를 찾기 위해 new\_node 의 score값인 ran\_num을 전달해 준다(이 경우 ran\_num 을 전달하지 않고 new\_node→score 값을 사용해도 된다.)

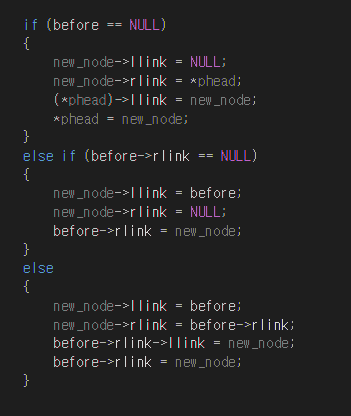
새로운 노드를 기존의 리스트에 삽입하기 위해서는 삽입되는 위치에 따라 여러가지의 경우를 살펴보아야 한다. 맨 앞에 삽입되는 경우, 맨 뒤에 삽입되는 경우, 그리고 중간에 삽입되는 경우. Dinsert\_node함수는 before 변수를 사용 해 삽입을 할 때 그 전의 노드를 체크하게 되는데, 각각의 경우마다 before값이 달라지게 된다.

|  |  |
| --- | --- |
| 삽입되는 위치 | Before값 |
| 맨 앞 | NULL |
| 맨 뒤 | 해당 위치의 전 노드, rlink = NULL |
| 중간 | 해당 위치의 전 노드 |

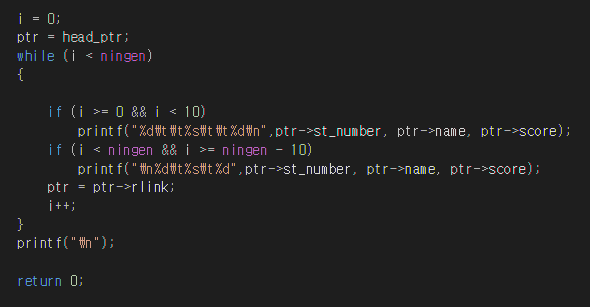


위의 경우로 나누면 코드를 다음과 같이 짤 수 있다. Ran\_num보다 큰 score값이 나올 때까지 p=p→rlink를 해 준다. 만약 해당 값이 phead일 경우(해당 노드가 맨 앞이 되는 경우) before = NULL, 아닐 경우(일반적인 경우) before = NULL을 해 준다. While 문의 조건으로 걸러지는 경우는 노드가 1개 있는 경우와 맨 뒤에 삽입하는 경우이다. 첫번째의 경우 1개의 노드와 비교를 하여 그 앞이면 before = NULL, 뒤면 before = p를 해 주는 것이고 맨 뒤일 경우 해당 위치 앞의 노드를 주는 것이다.

Before 값을 구하면 해당 값에 따라 삽입을 다르게 해 주어야 한다.

 before == NULL인 경우(맨 앞인 경우) 노드 삽입과 함께 \*phead의 값을 변경해 준다. Before→rlink == NULL 인 경우(맨 뒤인 경우) NULL값의 노드를 제외한 부분을 연결해 주고 나머지의 경우는 모든 연결을 해 준다.

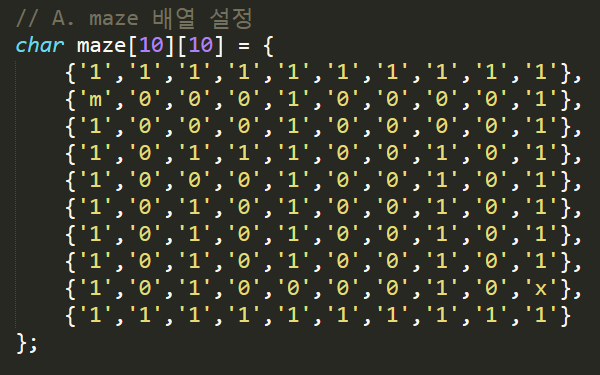
1. 상위 10명, 하위 10명의 학생 이름 정보 확인

위의 코드와 같이 head를 필두로 rlink로 옮겨가며 상위, 하위 10명씩을 출력하면 된다. 위의 경우는 학생 수를 알고있는 경우로 하였지만 학생 수를 모르는 경우 ptr = ptr→rlink를 하면서 NULL값이 되는 경우까지 찾으면 된다.

**[Prob#02] :** 미로 찾기

1. **Maze 배열 설정**

**;**문제에서 주어진 미로를 2차원 배열로 구현한 크기가 10\*10인 maze 배열이다.



막힌 길은 1, 뚫린 길은 0이며 입구인 (1, 0)을 m, 출구인 (8, 9)를 x 로 표현하였다.

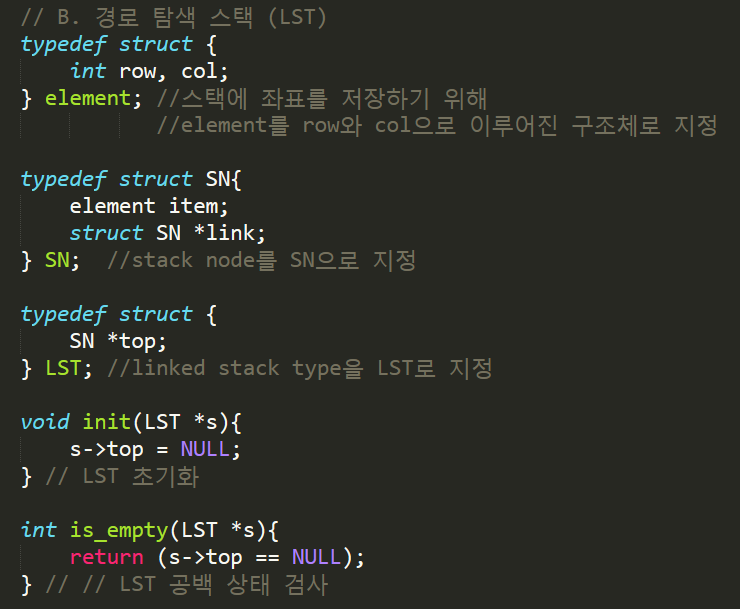
문제에서 maze를 int형으로 설정하라는 지시가 있었으나 m과 x가 int 형태가 아니므로 char형으로 구현하였다.

1. **경로 탐색**

경로 탐색을 위한 스택을 linked list로 구현하여 LST로 지칭하였다..

1. LST

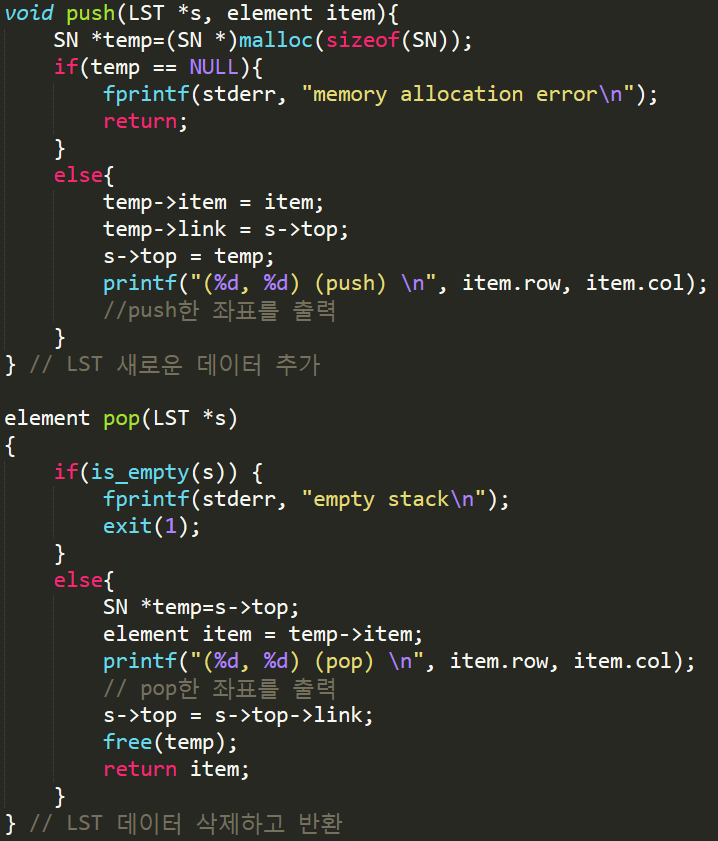
* typdef struct **element**: 좌표가 row와 column으로 이루어져 있으므로, element라는 구조체를 생성하여 좌표를 스택에 저장할 수 있는 형태로 만들었다.



* typedef struct **SN**: 스택의 노드를 위한 구조체를 SN으로 지정한다
* typedef struct **LST**: linked stack type을 LST로 지정한다.
* void **init**(LST \*s): LST 스택 초기화를 위한 함수이다.
* int **is\_empty**(LST \*s): LST 스택의 공백 여부를 판단하기 위한 함수이다.

1. LST의 push, pop

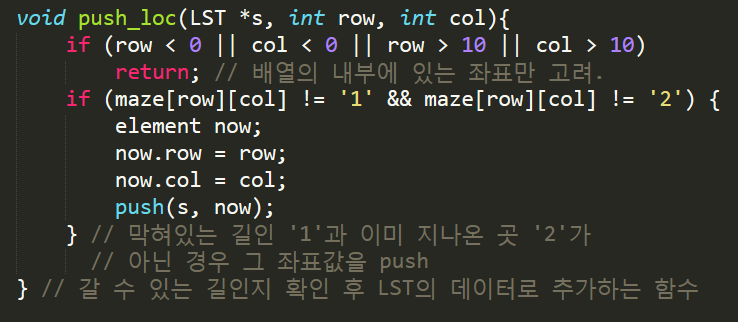
* (i) void **push**(LST \*s, element item): LST 스택에 새로운 데이터를 추가하기 위한 연산이다. 이 문제에서는 생쥐가 갈 수 있는 길을 저장하기 위해 사용된다. 더 이상 데이터를 추가할 공간이 없을 경우 에러를 반환한다. 아닐 경우 추가하려는 좌표를 새로운 노드 temp에 저장하고, temp의 link가 top을 가리키게 한 후 top이 temp를 가리키게 한다. 마지막으로 추가된 좌표인 push한 좌표를 출력한다.



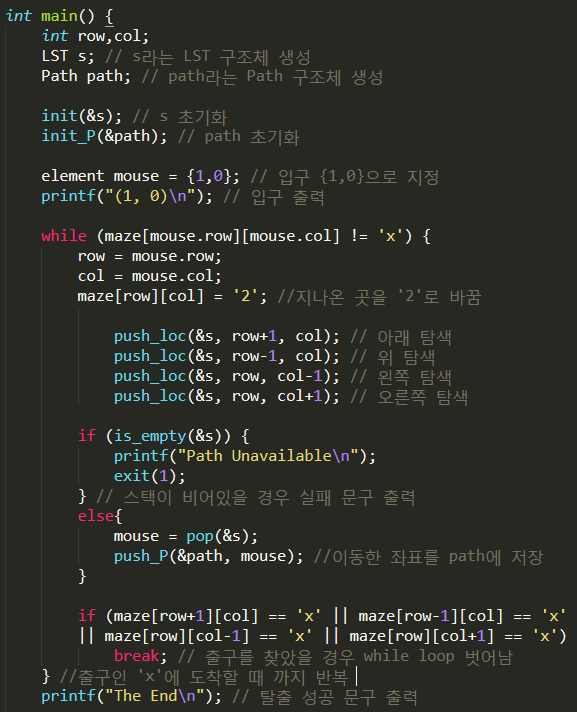
* (ii) element **pop**(LST \*s): LST 스택의 맨 위에 있는 요소를 반환하기 위한 연산이다. 이 문제에서는 생쥐가 움직인 좌표를 보이기 위해 사용된다. 더 이상 반환할 데이터가 없을 경우 에러를 출력한다. 아닐 경우 pop하는 좌표를 출력하여 생쥐가 움직인 좌표를 보이고, top을 (top-1) 위치의 노드를 가리키게 한 후 top이 가리키는 노드의 동적 메모리를 해제한다.

1. 가능한 경로 판단

* void **push\_loc**(LST \*s, int row, int col): 좌표가 배열의 내부에 있는지 판단하기 위해 첫번째 if문을 도입하였다. 만약 배열의 외부에 있는 경우 고려하지 않는다. 두번째 if문에서 생쥐가 움직일 수 있는 길인지 확인한다. 만약 ‘1’로 표현된 막혀있는 길이거나 ‘2’로 표현된 이미 지나온 곳인 경우 생쥐가 움직일 수 없기 때문에 그러한 경우가 아닐 때만 좌표값을 스택에 push한다. 이 함수를 이용하여 LST의 데이터가 될 수 있는지 여부를 판단한다.

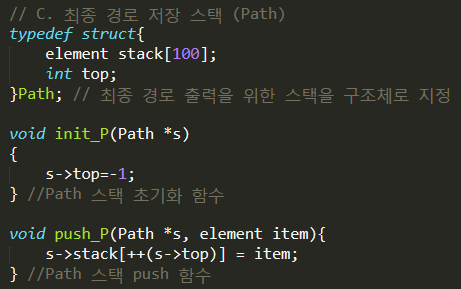
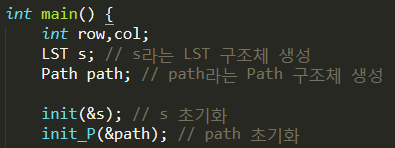


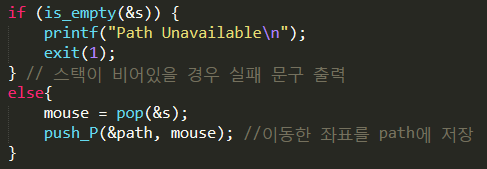
* main 함수에서는 움직일 수 있는 경로를 아래와 같이 판단한다. 여기서 등장하는 path 관련 부분들은 출력과 밀접한 관계가 있으므로 C. 에 설명을 적어두었다.
* 새로운 LST 구조체를 s로 지정하고, 초기화한다. 그 다음, 입구를 {1,0} 으로 지정한 후 입구를 출력한다.



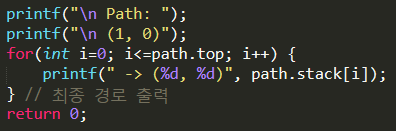
* while문: 생쥐가 출구에 도착하기 전까지 반복적으로 탐색을 진행한다. 출구가 아닐 경우, 생쥐가 지나온 곳을 ‘2’로 바꾸고, 아래, 위, 왼쪽, 오른쪽을 차례대로 탐색한다. 만약 스택이 비어있을 경우, 실패 문구를 출력한다. 그렇지 않을 경우 생쥐가 이동한 좌표를 pop한다. 만약 출구를 찾았을 경우, break를 통해 while loop를 벗어난다. 마지막으로 탈출 성공 문구를 출력한다.

1. **최종 경로 출력**

* 최종 경로 출력을 위한 스택을 Path라는 구조체로 지정한다. 이 스택에 LST에서 pop되는 좌표들, 즉, 생쥐가 움직이는 좌표들을 저장할 예정이므로 void **init\_P**(Path \*s)로 스택을 초기화하고 void **push\_P**(Path \*s, element item)으로 push 연산을 지정한다.
* main함수에서 path라는 Path 구조체를 생성하고, path를 초기화한다.
* **push\_P**(&path, mouse): 최종 경로를 저장할 수 있도록 생쥐가 이동한 좌표인 LST에서 pop된 좌표를 path에 push한다.



* for문을 사용해서 path에 저장된 좌표들을 연속적으로 출력한다. 이를 통해 최종 경로가 출력된다.



**[Prob#03] :**Queue Simulation

Znn.com의 Slashdot Effect를 해결하기 위한 시뮬레이션

시뮬레이션 시간: 100초 / 사용자 수: 20명

1. **자료구조**

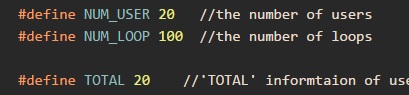
News contents와 Users’ Information크게 두 가지 자료구조를 가진다.

1. News contents: Queuenews\_queue

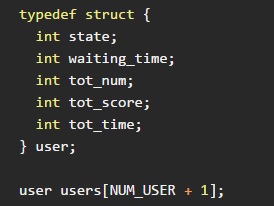
사용자 식별자(번호)와 fidelity의 구조체를 멤버 변수로 갖는 배열 기반 순환 큐이다.

1. Users’ information: Arrayusers

각 사용자에 대한 여러가지 정보를 담는 구조체 배열이다. 20명의 사용자 정보를 담는 공간 + 전체 사용자가 공유하는 데이터를 담는 1칸으로 배열의 크기는 21이다.

구조체의 멤버변수 목록과 각 변수가 나타내는 정보는 아래와 같다.

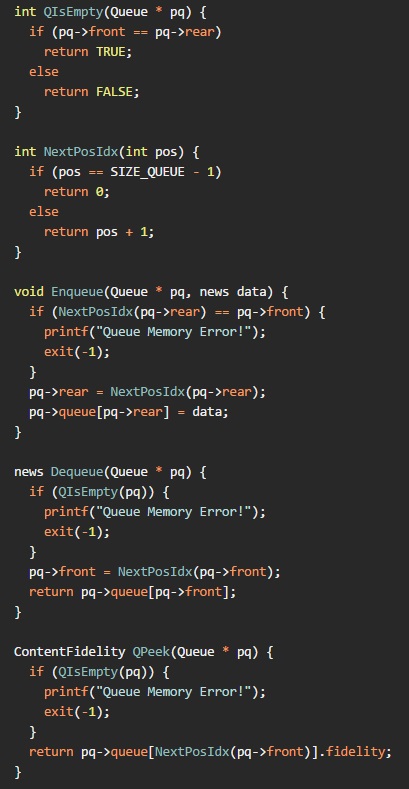
|  |  |  |
| --- | --- | --- |
|  | 20명의 사용자  index 0 ~ 19 | ‘TOTAL’ info  index 20 (TOTAL) |
| state | 0: request 가능  1: 컨텐츠 대기중  request 불가능 | X |
| waiting\_time | 현재 request한 컨텐츠에 대한 대기 시간 | X |
| tot\_num | request했던 컨텐츠 수 (누적) | 현재 대기중인 사용자 수 |
| tot\_score | request했던 컨텐츠들의 점수 합 (누적) | X |
| tot\_time | request했던 컨텐츠들을 제공받기까지 대기시간 합 (누적) | 현재 대기중인 사용자들의 대기 시간 합 |



tot\_num,tot\_time은 index 0~9에서, 그리고 index 20에서 의미하는 바가 크게 다르다.

1. **Queue의 ADT와 구현**

Queue에 대해 총 5가지의 기능을 구현하였다.

* Int **QIsEmpty** (Queue \* pq)

현재 큐가 비어있는지 확인한다.비어있다면 1, 아니면 0 리턴

* Int **NextPosIdx** (int pos)

순환큐이기 때문에 현재 위치의 다음 칸을 해당 함수를 이용해 계산한다.

* void **Enqueue** (Queue \* pq, news data)

큐에 데이터(news: 사용자 식별자와 fidelity의 pair)를 저장한다.

* News **Dequeue** (Queue \* pq)

가장 앞선 데이터를 삭제하고 반환한다.

* ContentFidelity **QPeek** (Queue \* pq)

가장 앞선 데이터를 삭제하지 않고 데이터 정보만 추출하는 함수이다. 문제에서 필요한 정보는 fidelity이기 때문에 반환형은 ContentFidelity이다.

1. **main함수의 실행흐름과 데이터 값의 변화 시점**

main함수에선 for문을 총 100번 (100초를 의미) 돌면서 4가지 함수를 반복하여 실행한다.

* 매 초마다 printQueue() -> provide()
* 2초 마다 setFidelity() -> request() -> printQueue() -> provide()

를 차례대로 호출한다.

그리고 100번의 반복문이 끝나면 printResult()함수를 호출하며 실행을 마무리 한다.

시뮬레이션을 진행하면서 얽혀있는 여러 데이터들의 값을 변경시켜야 하는데 각 데이터의 변경시점은 다르다. Enqueue와 동시에, 또는 Dequeue와 동시에 변화시켜야 하는 값이 있는 반면에 매 초마다 값을 증가시켜야 하는 데이터도 있다.또한 각 데이터들의 사용 시점 (참조 시점)도 다른데 자세한 내용은 아래 표와 같다.표에서 실행 흐름은 위에서 아래로 진행된다.

(printQueue함수가 1초에 한 번씩 호출되므로 매 초마다 해야할 작업들은 이 함수 내에 구현하였다.)

|  |  |  |  |
| --- | --- | --- | --- |
| 2초에 한 번씩 | 1초 마다 | 각 사용자 데이터 (User) | 사용자들이 공유하는 데이터 (TOTAL) |
| **setFidelity** |  |  | **[사용]**  사용자의 총 대기시간(tot\_time),사용자의 수(tot\_num)를 이용해 평균 대기시간을 계산 |
| **request**  (Enqueue와 동시에) |  | **[값 변경]**  state 1로 변경  waiting time **-1**로 setting | **[값 변경]**  사용자 수 (tot\_num) 1 증가 |
|  | **printQueue**  (매 초 마다 해야하는 작업) | **[값 변경]**  waiting\_time 1초 증가 | **[값 변경]**  대기 중인 사용자 수 만큼 사용자 총 대기시간 (tot\_time) 증가 |
|  | **Provide** (Dequeue와 동시에) | **[사용]**  waiting\_time이 tot\_time에 더해지고 사용자들의 대기시간 총 합에서 제외됨 |  |
| **[값 변경]**  tot\_time에 컨텐츠의 waiting\_time이 누적됨  tot\_num이 1 증가 (컨텐츠 수 누적 증가)  tot\_score에 컨텐츠 점수 누적  state 다시 0으로 변경 | **[값 변경]**  tot\_time  사용자들의 (current) 대기시간에서 (제공된 컨텐츠의 대기시간 **+ 1**)을 뺌  tot\_num  대기중인 사용자 수를감소시킴 |

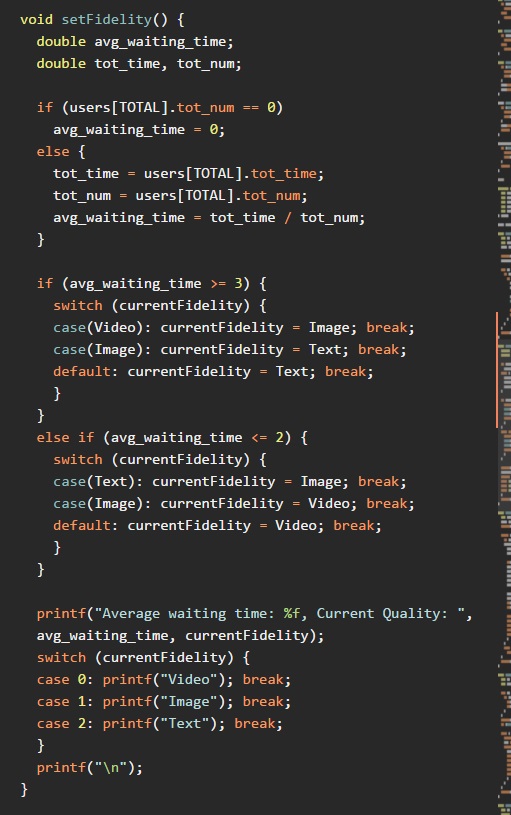
provide함수에서 TOTAL의 tot\_time값 (사용자들의 총 대기시간)을 변경할 때 제공되는 컨텐츠의 대기시간을 빼고 1을 더 빼주고 있는데 그 이유는 TOTAL 데이터와 사용자 데이터의 사용시점이 다르기 때문이다.

데이터가 들어옴과 동시에 제공되는 상황을 생각해보자.

매 초마다 각 사용자 데이터에서 waiting\_time을 1 증가 시켜줘야 하는데 이는 printQueue에서 진행되고 해당 데이터는 바로 다음에 실행되는 provide에서 사용된다.따라서 0초를 표현하기 위해서는 printQueue이전에 Enqueue되는 시점에서 waiting\_time을 -1로 설정해줘야 한다.

TOTAL데이터도 마찬가지로 매 초마다 대기시간을 대기자 수만큼 증가시켜줘야 하고 printQueue에서 진행된다. 하지만 사용자 데이터와 다르게 TOTAL 데이터는 printQueue이전인 setFidelity함수에서 사용된다.매 초에 대한 정보를 업데이트 하기 전에 사용되기 때문에 TOTAL 데이터는 실제보다 1초 더 늘려서 값을 저장한다.따라서 제공된 컨텐츠의 대기시간을 제외시킬 땐 더 늘어난 1초까지 합해서 감소시켜야 한다.

1. **함수 구현**



**setFidelity**

TOTAL데이터의 tot\_time (사용자들의 대기시간 총 합)과 tot\_num (대기 중인 사용자들의 수)를 이용해 사용자들의 평균 대기 시간을 계산한다.

그 값이 3 이상이라면 contemts의 fidelity를 한 단계씩 낮추고

2 이하라면 반대로 한 단계씩 높인다.

그리고 현재 평균 대기시간과 Fidelity를 출력한다.

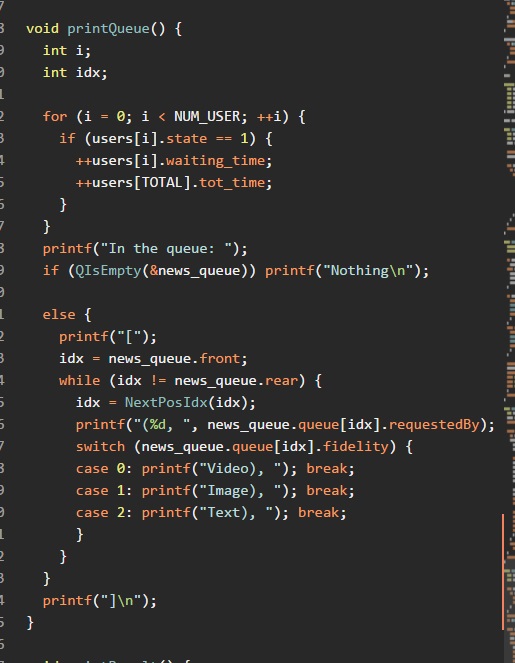
**request**

요청 가능한 상태인 사용자들에 대해 50%확률로 컨텐츠를 요청한다.

컨텐츠를 요청한 사용자의 state는 1로, waiting\_time은 -1로 setting된다.

위에서 계산된 currentFidelity에 따라 사용자 식별자와 함께 컨텐츠 대기 큐로 들어가게 된다.

(**Enqueue)**

****

**printQueue**

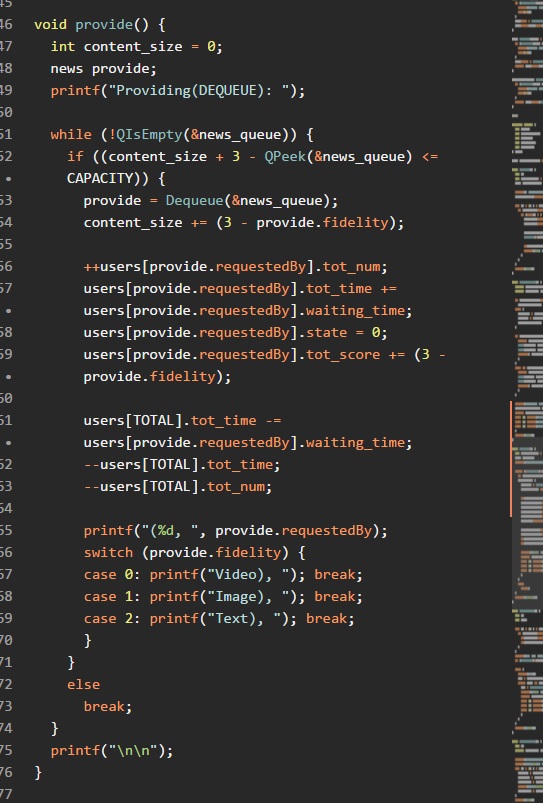
매 초마다 수행되어야 하는 작업들이 여기서 진행된다.

for문을 돌리면서 state가 1인 컨텐츠 요청 후 대기중인 사용자들에 대해서

해당 사용자 데이터의 waiting\_time을 1 증가시키고

TOTAL 데이터의 tot\_time을 if문을 통과할 때마다 증가시킨다. (대기 중인 사용자 수 만큼)

그리고 front 부터 rear까지 순환큐를 돌면서 데이터(news)들을 출력한다.

**provide**

CAPACITY만큼 데이터를 제공하는 함수이다.

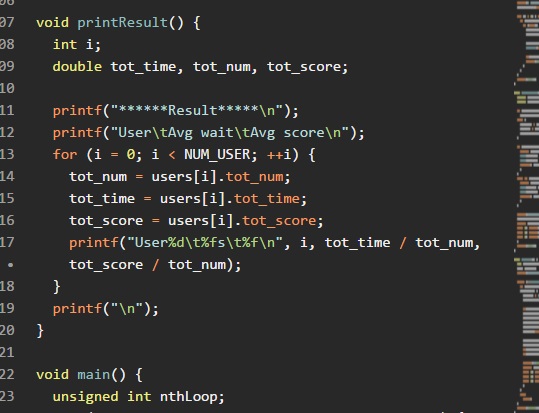
순환큐가 비어 있지 않을 경우에 한해서

순환큐에서 다음 꺼낼 데이터의 fidelity가 누적될 경우 CAPACITY를 초과하지 않을 크기라면 **Dequeue**를 진행한다.

이때 데이터를 꺼내도 되는지 여부를 체크할 때 QPeek함수가 사용된다.

그리고 위의 표에서 설명했 듯 사용자 데이터의 변수 값들을 변경시키고 TOTAL 데이터 값들도 변경시킨다.

마지막으로 제공된 컨텐츠 정보를 출력하고 함수를 종료시킨다.

**printResult**

사용자 각각에 대해 평균 대기시간과 평균 컨텐츠 점수를 출력한다.

사용자 데이터에 누적 데이터들 (tot\_time, tot\_num, tot\_score)을 포함한 이유이다.

double형으로 데이터를 다시 저장한 후 계산하여 소수점까지 출력한다.