Homework 1

**<ADT>**

|  |  |
| --- | --- |
| **LinkedStackType 구조체** | |
| **data type** | **name** |
| DlistNode\* | top |
| DlistNode\* | head |

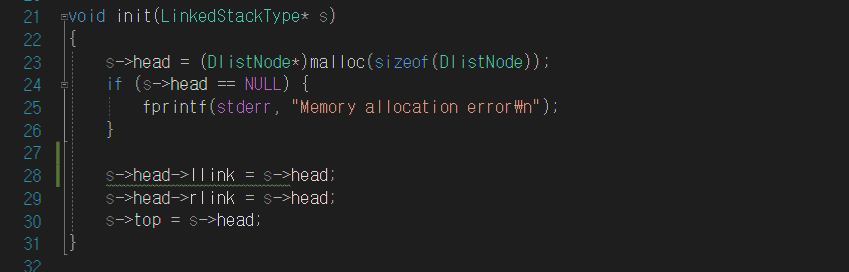
**Variables**

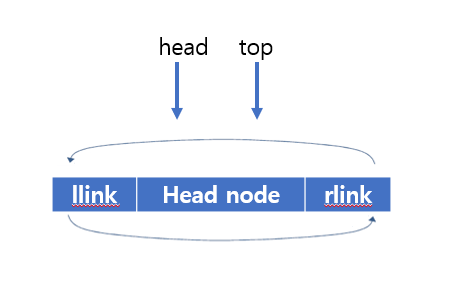
|  |  |
| --- | --- |
| **DlistNode 구조체** | |
| **data type** | **name** |
| element (=int) | data |
| DlistNode\* | rlink |
| DlistNode\* | llink |

문제 해결을 위해 Doubly linked list의 Stack을 구현할 때, Doubly linked list에 편리하게 접근하기 위하여 head node를 두는 방식을 선택했다. (data 노드의 양 끝을 잇는 방법으로 DlistNode\* top 만을 이용하여 구현할 수도 있지만) 이 때, LinkedStackType 의 스택 변수를 통해서 **top**과 **head node**에 접근할 수 있도록 하기 위해서 stack 구조체 안에 top과 head를 DlistNode의 주소 값을 기억할 수 있는 포인터 형태의 구조체 멤버로 만들어 놓았다.

**Functions**

1. void init(LinkedStackType\* s)



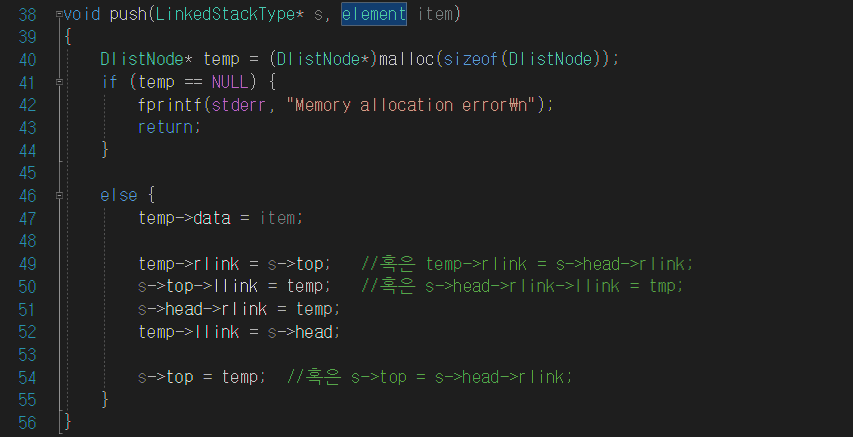
init( ) 함수를 통해 stack을 초기화 하고 나면, 왼쪽과 같은 형태로 stack이 초기화된다.

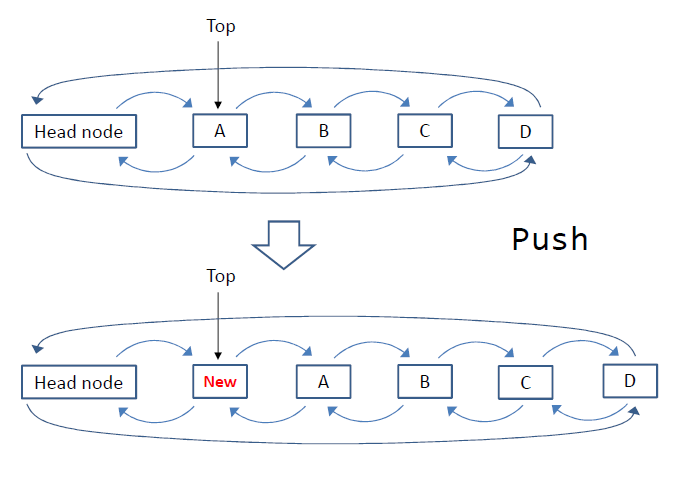
1. int is\_empty(LinkedStackType\* s)



**1)**의 stack 초기화 함수에서 살펴보았듯, 초기화 한 후에 어떤 노드도 push 되지 않은 상태에서는 top과 head 포인터 모두 head node를 가리키고 있게 되므로, 두 포인터의 주소 값이 같다면 stack이 empty 상태임을 알 수 있다.

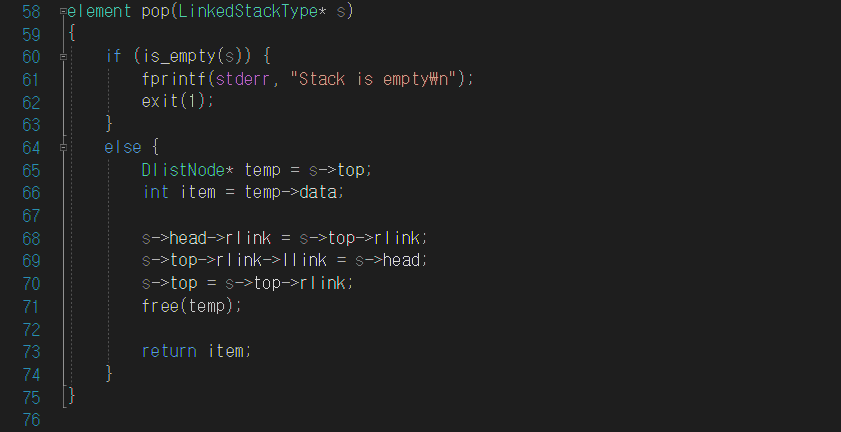
1. void push(LinkedStackType\* s, element item)

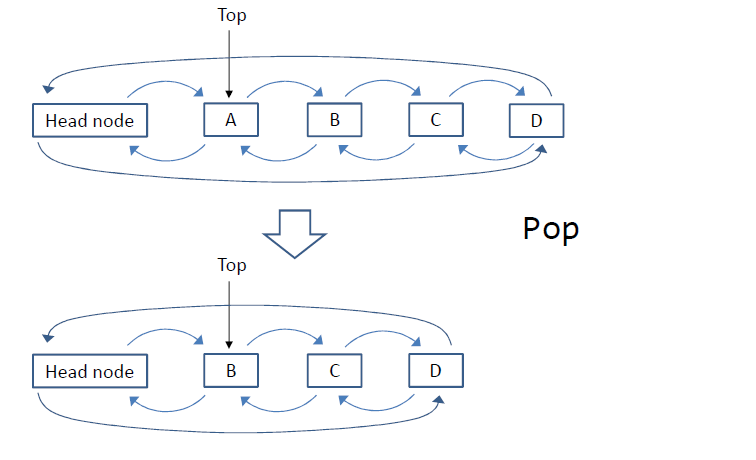




**Doubly linked list로 구현된 Linked Stack에 새로운 노드를 맨 앞에 Push 하기 위한 절차**

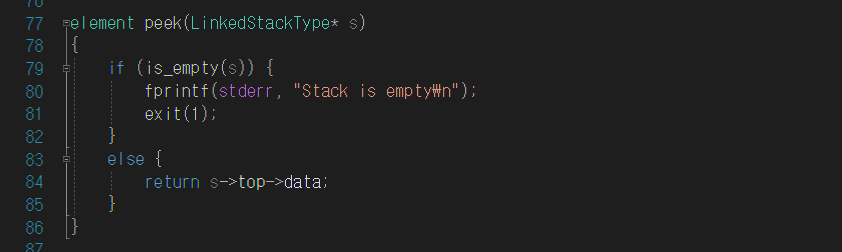
1. 새로운 노드 temp를 만들어 data 값을 저장한다.
2. temp의 오른쪽 링크를 A와 연결
3. A의 왼쪽 링크를 temp와 연결
4. Head node의 오른쪽 링크를 temp와 연결
5. temp의 왼쪽 링크를 head node와 연결
6. top의 포인터 주소 값을 새로운 노드 temp로 업데이트
7. element pop(LinkedStackType\* s)





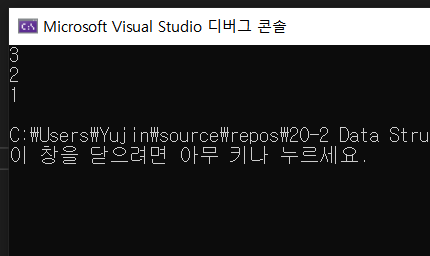
**Doubly linked list로 구현된 Linked Stack에서 맨 앞의 노드를 Pop 하기 위한 절차**

1. 새로운 구조체 포인터 변수 temp를 만들어 삭제할 노드의 주소값을 저장한다.
2. 리턴할 item 값을 저장한다.
3. Head node의 오른쪽 링크를 B와 연결
4. B의 왼쪽 링크를 head node와 연결
5. top의 포인터 주소 값을 B로 업데이트
6. 메모리 해제
7. 제거한 노드의 data 값 반환
8. element peek(LinkedStackType\* s)



스택의 가장 위에 있는 노드(=첫번째 노드)를 삭제하지 않고 data 값만 반환한다.

**실행결과**

****

Homework 2

**문제해결방법**

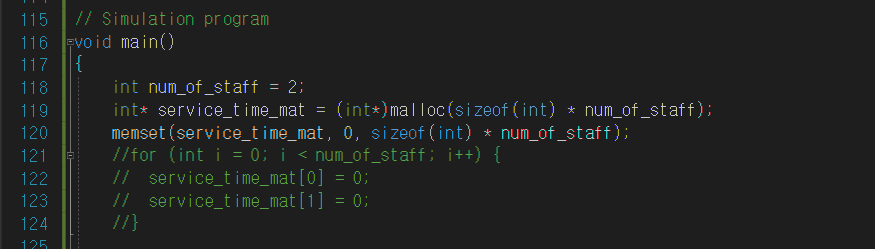
Bank에 staff가 1명 -> 2명으로 늘어날 때 수정해 줘야 하는 점을 생각해보자.

직원이 1명 뿐일 때는 1) 직원이 서비스 중이거나 2) 직원이 서비스 중이지 않거나 두가지 경우만 고려해주면 된다. 하지만 직원이 2명 (0번직원과 1번직원)으로 늘어난다면 1) 직원 두 명이 모두 서비스 중인 경우 2) 0번 직원만 서비스 중이고, 1번 직원은 서비스 중이지 않은 경우 3) 0번 직원은 서비스 중이지 않고, 1번 직원은 서비스 중인 경우 4) 직원 두 명이 모두 서비스 중이지 않은 경우 로 네가지 경우를 고려해 줘야 한다.

따라서 네 가지 경우의 수를 모두 고려하여, main의 코드를 수정해준다.

**Code Explanation**

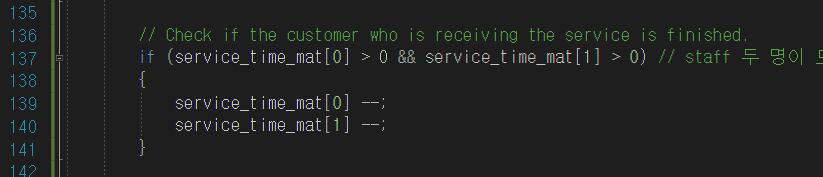
1. 초기 조건 설정



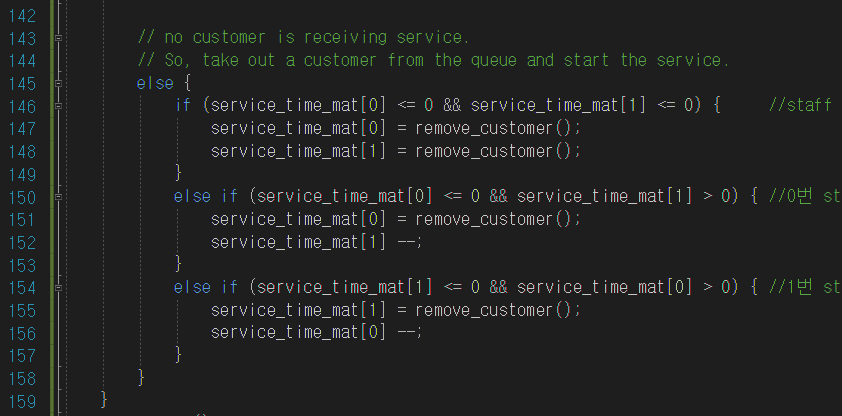
Staff의 인원수를 2명으로 설정하고, 두 명의 staff의 service time을 각각 저장할 int형 포인터 변수를 위한 메모리 공간을 할당한 후, memset함수를 이용해 모두 0으로 초기화한다.

Simulation time을 10으로 설정한 것은 동일하며, clock이 1 증가할 때마다 다음과 같은 조건을 체크합니다.

1. staff 두 명이 모두 서비스 중일 때



Service\_time\_mat 변수에는 현재 서비스 중인 고객의 남은 서비스 시간이 저장되어 있다. (정확히는 int 값이 저장되어 있는 주소를 가지고 있는 포인터) 두명의 staff의 남은 service 시간을 확인하여 둘다 0보다 크다면 1씩 감소시킨 후, dequeue 작업(다음 손님을 받는 작업) 없이 다음 clock으로 넘어간다. 반대로 이 경우가 아니라면, 두 staff 중 적어도 한 명의 staff가 서비스 하고 있지 않다는 말이 되므로, 적어도 1번 이상 dequeue 작업이 수행되어야 한다.



1. staff 두 명이 모두 서비스가 끝났을 때

: remove\_customer() 함수를 통해서 queue에서 대기하고 있는 고객을 꺼내 서비스를 시작한다. (이 때 변수에는 고객이 필요한 service time이 담기게 된다.) 두 명의 staff 모두 서비스 하고 있지 않은 상태이므로 remove\_customer()를 두번 진행한다.

1. 0번 staff만 서비스가 끝났을 때

: 0번 staff에는 remove\_customer()를 통해 새로운 고객을 queue에서 꺼내주고, 1번 staff는 아직 서비스 중이므로 dequeue하지 않고 service\_time 변수를 1 감소시킨다.

1. 1번 staff만 서비스가 끝났을 때

: 반대로 이번에는 1번 staff에는 remove\_customer()를 통해 새로운 고객을 queue에서 꺼내주고, 0번 staff는 아직 서비스 중이므로 dequeue하지 않고 service\_time 변수를 1 감소시킨다.

**실행결과**

**Time 1 2 3 4 5 6 7 8 9 10**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **staff[0]** | **0** | **0** | **0** | **2** | **2** | **2** | **3** | **3** | **3** | **3** |
| **staff[1]** |  | **1** | **1** | **1** | **1** | **1** | **4** | **4** | **4** | **4** |

**고객 0번 도착 2번 도착 3번 도착**

**1번 도착 4번 도착**

