REPORT

Data Structures

Student ID : 1771008
Name : Minjeong Kim

Homework4_1

변수(varialbe) 분석

LinkedStackType	
type	**name**
DlistNode *	top

DlistNode	
type	**name**
element	data
DlistNode *	rlink
DlistNode *	llink

main 함수에서 DListNode와 LinkedStackType을 할당하면 다음과 같은 자료형이 형성된다.
이때, LinkedStackType하나가 stack하나를 의미하고, DlistNode는 stack에 쌓이는 자료를 의미한다.
즉, LinkedStackType하나당 DlistNode는 여러 개 존재한다.

함수(function) 분석

1. init(LinkedStackType *s)

main함수에서 init을 하게되면, s->top이 NULL로 설정된다. 이때 s->top의 의미는 s스택의 top을 가리키는 포인터인데, init함수를 통해서 포인터만 생성하고, 아직은 가리키는 Node값은 존재하지 않는다.

2. is_empty(LinkedStackType *s)

init상태와 같이 s->top==NULL일 때 is_empty()함수는 true가 된다.
s->top!=NULL 일 때, 즉 포인터가 가리키는 Node값이 있을 때는 false가 된다.

3. push(LinkedStackType *s, element data)

일단 stack의 특성 **FILO**을 고려하기 위해 i/o가 들어오는 방향을 먼저 정해줘야한다.
본인은 stack의 top을 오른쪽으로, bottom을 왼쪽으로 생각했다. 따라서 push를 할 때, 새로운 노드는
기존 node의 llink와 연결됨을 주의하자!

ㄱ. is_empty(s) 상태에서 push할 때
s->top=NULL 상태이므로, 일반적인 경우처럼 push를 하면, llink, rlink연결이 안되있어서 linked list구현이 안된다.
따라서 is_empty(s)인 경우를 나눠서 생각해야한다.
circle doubly linked list임을 생각하면, 처음 node를 삽입 했을 때 rlink, llink 모두 자기 자신을 가리켜야 한다.

ㄴ. 일반적인 경우에서 push할 때,
강의 노트4에 있는 doubly linked list의 insert와 같다. s->top을 before로 생각하고 temp를 new_node로 생각하면 같은 코드임을 알 수 있다.

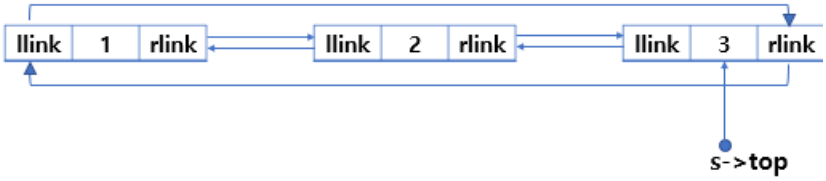
4. pop(LinkedStackType *s)

ㄱ. s->top == s->top->llink경우에서 pop할 때
node가 하나 남았을 때 pop을 하면 init과 같은 상태인 p->top==NULL 상태로 돌아가야 하기 때문에 나눠서 생각한다. node가 하나 남았을 땐, 노드의 llink, rlink가 자기자신을 가리키기 때문에 llink가 가리키는 노드가 자기 자신인지를 확인한다.
이때! llink == rlink를 하면 안된다. node가 두개 남았을 때에도 p->top->llink == p->top->rlink이므로
link와 node자신이 같은지로 확인해야한다.

ㄴ. 일반적인 경우에서 pop할 때,
강의 노트4에 있는 doubly linked list의 delete와 같다. temp를 removed로 생각하면 같은 코드임을 알 수 있다.
이때 pop을 하는 것은 node를 꺼내는(=지우는) 것이므로 메모리 할당을 해제해주는 것을 잊지 말자.
pop의 경우 top의 node를 꺼내면서 node의 data값을 return하므로 return값을 위한 top의 data값을 할당 해제하기 전에 새로운 변수에 저장해 둔다.

5. peek(LinkedStackType *s)
top에 있는 요소의 data값을 return한다.

main함수에서 push까지 했을 때를 그림으로 표현해 둔 것이다. s->top이 node를 가리키는 포인터라는 점과 circular doubly linked list이므로 top의 rlink는 bottom을, bottom의 llink는 top을 가리킴을 기억하자.



Homework4_2

변수(variable) 분석

element	
type	**name**
int	id
int	arrival_time
int	service_time

QueueType	
type	**name**
element[MQS]	queue
int	front
int	rear

전역변수		
type	**name**	**Description**
QueueType	queue	은행창구 대기 줄
int	duration = 10	서비스 제공 시간 (10분)
double	arrival_prob = 0.7	한 타임 당 고객이 도착할 확률
int	max_serv_time = 5	한 고객 당 은행업무 최대 시간
int	clock	while문으로 현재 시간 체크용도
int	customers	고객의 총 숫자
int	served_customers	업무를 본 고객의 총 숫자
int	waited_time	고객이 기다린 시간의 합
main함수 지역변수		
int	service_time	현재 창구가 비기까지 남은 시간

함수(function) 분석

[큐 구현 함수]

본 과제에서 큐의 형태는 circular임을 명심하자, 이때 큐의 사이즈는 MAX_QUEUE_SIZE이다.

1.is_empty(QueueType *q)

circular 큐에서 비었을 때는 front와 rear가 같을 때이다.

2. is_full(QueueType *q)

circular 큐에서 큐가 가득 찼을 때는 front가 rear의 1칸 앞에 있을 때이다.
circular 이므로 %기호를 이용해서 rear와 front값을 확인한다.

3. enqueue(QueueType *q, element item)

queue에 데이터가 들어갈 때는 rear의 위치가 1 증가한다. (circular이므로 % 연산을 잊지말자)
stack은 데이터를 넣고나서 top의 위치를 옮겼다면, queue는 반대로 rear의 위치를 옮긴 후에, 데이터를 넣는다.

4. dequeue(QueueType *q)

queue에서 데이터를 뺄 때는 front의 위치가 1 증가한다. (circular이므로 % 연산을 잊지말자)
enqueue와 마찬가지로, dequeue도 front의 위치를 옮긴 후에, 데이터를 뺀다.

[시뮬레이션 구현 함수]

1.is_customer_arrived()

arrival_prob를 적용시키기 위한 함수이다. 실제 상황에서 매 분마다 손님이 오는 것이 아니기 때문에, 분당 손님이 올 확률을 arrival_prob로 설정하고 random()함수를 이용해서 확률을 구체화한다.

2. insert_customer(int arrival_time)

인자로 받는 arrival_time은 메인함수의 clock에 해당한다. (clock = 시뮬레이션 시작부터 매 time 카운트)
기본적으로는 queue에 customer의 정보를 넣는 역할을 하는 함수이다. 이때 customer의 정보는 위에서 정의한 구조체 element 타입형태로 저장한다. 따라서 id, arrival_time, service_time값을 저장한다.
이때 max_serv_time = 5로 설정했기 때문에, service time은 1이상 5미만으로 구현해야하는데, 이를 위해 random() 함수를 사용했다. random()함수는 [0.0 , 1.0) 범위중 임의의 수를 return하는 함수이다. 따라서
(int)(5*random())+1을 이용해서 1이상 5이하의 임의의 수를 return한다.

insert_customer() 함수의 기능은 은행창구의 대기 줄을 만드는 역할이다. 즉, 고객은 insert_customer()함수로 대기줄 안에 들어가게 되는 것인데, 이때, 고객이 은행에 도착한 시각(arrival_time), 은행을 이용할 시간(service_time), 고객의 임의 번호 (id) 의 정보를 갖고 대기줄에서 고객이 기다리고 있다고 생각하자.

3. remove_customer()

remove_customer()은 main함수에서 while문안에 있는 if-else문의 조건이 만족되었을 때 실행된다.
먼저 insert_customer()로 queue에 고객이 대기줄 안에 위치하게 되고, main함수안에있는 전역변수 service_time에 따라 고객이 대기줄을 나가 (remove_customer) 은행창구에서 업무를 볼지 여부가 정해진다. service_time의 의미는 현재 창구에서 일이 끝나기까지 남은 시간을 의미한다. 따라서 service_time이 0이라면 대기줄에 있던 고객이 창구에서 업무를 봐야하기 때문에, remove_customer()함수를 실행시키고, queue에서 뺀다.

service_time [지역] : main함수에 있는 창구가 비기까지 남은시간인 service_time에 값을 저장하므로,
element의 service_time 멤버의 값을 불러와 -1을 하고 main함수로 return한다.
served_customers [전역] : 응대한 고객의 수가 하나 늘었으므로 ++ 연산을 한다.
waited_time [전역] : 고객이 기다린 시간의 합이므로 element의 arrival_time멤버와 clock의 값을 연산해 합한다.

4. print_stat()

main()함수의 duration동안의 반복과 customer의 inser, remove함수의 반복의 결과 누적된 전역변수들의 값을 이용하여 최종 결과를 출력한다.

5.main()함수

main()함수의 while 반복문에 따라 duration크기만큼 반복한다.

매 반복마다 is_customer_arrived()함수를 이용해서 고객의 도착여부를 확인하고 도착했다면, 전역변수 queue에 고객에 관한 정보를 넣는다.

service_time은 은행창구가 비기까지 남은 시간을 의미하는데, 남은 시간이 0이되면 대기줄에 있던 고객이 창구에 업무를 보러와야하기 때문에 remove_customer()함수를 실행하고, 그결과 return되는 service_time을 할당한다.
while 반복마다 남은 service_time이 0이 아니면 대기줄에 그대로 고객을 대기해야하므로 remove_customer() 함수를 실행하지 않고, 은행창구가 비기까지 남은시간을 줄인다.

문제 분석

주어진 코드는 bank_staff가 한 명일 때이고, 원하는 해답은 bank_staff가 두 명일 때이다.

이때 문제의 핵심은 queue의 remove이다.
은행창구가 비면 대기줄(queue)에 있던 손님이 빈 창구로 가기 때문에, 은행창구가 2개라는 소리는 remove와 관련된 연산이 1개였던 것을 2개로 늘리라는 소리이기 때문이다.

따라서 한 창구가 비기까지 남은 시간인 service_time의 수를 2개로 늘리고(service_time1, service_time2),
매 반복마다 service_time[0]과 service_time[1] 모두 접근해서 remove_customer()를 해야하는지 확인하면
결국 bank_staff가 두 명일때와 같은 양상을 보인다.

본인은 고객이 어느 창구로 갔는지까지 확인하기위해 revmoe_customer()안에 인자를 넣어서 함수안 printf문에서 창구 번호를 출력하였다.

문제 확장

bank_staff를 늘리기위해서는 main()의 service_time의 개수만 늘려주면 된다는 것을 알았다.

따라서, 문제를 확장하여 bank_staff 수 등 다양한 변수를 설정하여, 은행창구 시뮬레이션의 결과를 확인하자

시뮬레이션 상황에 대한 변수를 input값으로 주고 시뮬레이션 사용자가 값을 조정할 수 있게 한다.
은행창구 현장의 다양한 상황에 대한 시뮬레이션을 할 수 있도록 코드를 수정하였다.
먼저 service_time을 사용자가 입력한 수로 배열의 크기를 설정한다.
또한 시뮬레이션 시간, 고객 도착확률, 업무최대시간 등 지역변수도 사용자가 입력한 수로 설정한다.

수정한 변수는 다음과 같다.

1. 창구의 수(=bank_staff의 수) // int bank_staffs_number
2. 한 타임에 고객이 올 확률 (=>n분당 1명일 경우엔 1/n을 대입) //float arrival_prob
3. 시뮬레이션 시간 (=n분) //int duration
4. 한명이 은행창구에서 업무를 보는 최대 시간 //max_serv_time

시뮬레이션의 결과가 유의미하기 위해서는 duration의 숫자가 커야 한다.
빈약하지만 몇 개의 결과로 은행 시뮬레이션을 시작해보자.

	입력				출력
	bank_staffs	arrival_prob	duration	max_serv_time	
1	2	0.7	10	5	Number of customers served = 5 Total wait time = 4 minutes Average wait time per person = 0.800000 minutes Number of customers still waiting = 2
2	2	0.7	100	5	Number of customers served = 65 Total wait time = 89 minutes Average wait time per person = 1.369231 minutes Number of customers still waiting = 2
3	2	0.7	500	5	Number of customers served = 335 Total wait time = 4047 minutes Average wait time per person = 12.080597 minutes Number of customers still waiting = 9
4	2	0.7	1000	5	Number of customers served = 664 Total wait time = 12048 minutes Average wait time per person = 18.144578 minutes Number of customers still waiting = 32
5	3	0.7	500	5	Number of customers served = 344 Total wait time = 47 minutes Average wait time per person = 0.136628 minutes Number of customers still waiting = 0
6	3	0.7	500	10	Number of customers served = 280 Total wait time = 12931 minutes Average wait time per person = 46.182143 minutes Number of customers still waiting = 64
7	4	0.3	500	20	Number of customers served = 151 Total wait time = 691 minutes Average wait time per person = 4.576159 minutes Number of customers still waiting = 0

1~4 : duration이 늘어날수록 average가 늘어난다. (하지만, 수요에 비해 공급이 적기 때문에 일어나는 현상 가능)
따라서 500분으로 가정 (8시간 : 하루 은행 업무시간)

5~6: 1인 은행업무시간 늘어날수록 average 늘어난다.
1인 업무시간 더 증가

7: 부스가 4개일 때 시뮬레이션

유의미한 시뮬레이션 결과를 얻기 위해서는 결과 데이터를 분석하는 또다른 소스 혹은 툴이 필요함을 느낌