1. 개요

리눅스 환경에서 one line으로 입력을 받아 stack이나 queue에 저장하고 바로

output 시키는 프로그램을 코딩해보면서 리눅스 환경에 적응해보도록 한다.

2. 프로그램 구조 설명

2.1 함수에 대한 설명

-SQApi.c

//stack부분

1. S\_Head\* MakeStack()

Stack head를 만들어서 초기값 설정 한 뒤 주소값을 리턴한다.

2. int PushStack(S\_Head\* stack,void\* data)

Stack에 새로운 값을 넣는다. 성공하면 1, 실패하면 0을 리턴한다.

3. void\* PopStack(S\_Head\* stack)

Stack에서 가장 마지막으로 넣은 값을 꺼낸다. 꺼낸 데이터의 주소값을 리턴한다.

//queue부분

4. Q\_Head\* MakeQueue(){

Queue head를 만들어서 초기값 설정 한 뒤 주소값을 리턴한다.

5. int Enqueue(Q\_Head\* head, void\* data)

Queue에 새로운 값을 넣는다. 성공하면 1, 실패하면 0을 리턴한다.

6. void\* Dequeue(Q\_Head\* head)

Queue에서 가장 처음 넣은 값을 꺼낸다. 꺼낸 데이터의 주소값을 리턴한다.

-SQmain.c

1. void SQ\_InputFunction(void\* SQ,int type)

type이 0이면 void\* 타입 인자를 stack으로 인식, type이 1이면 queue로 인식한다.

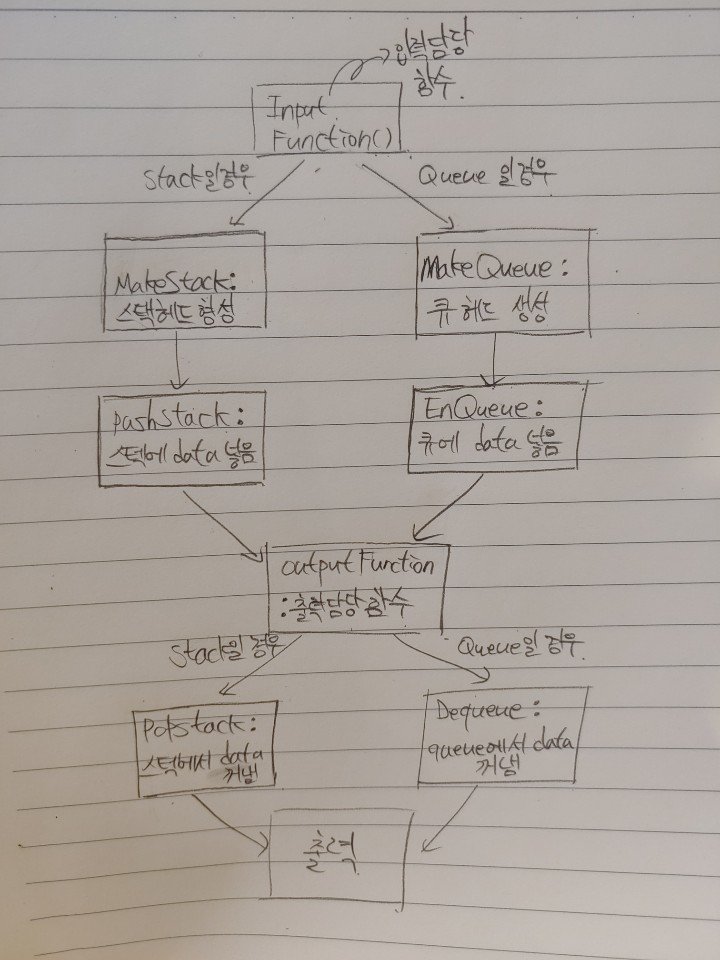
인식 후에 stack이나 queue에 one line으로 입력을 받아서 자료를 넣는다.

2. void SQ\_OutputFunction(void\* SQ,int type)

Type이 0이면 void\* 타입 인자를 stack으로 인식, type이 1이면 queue로 인식한다.

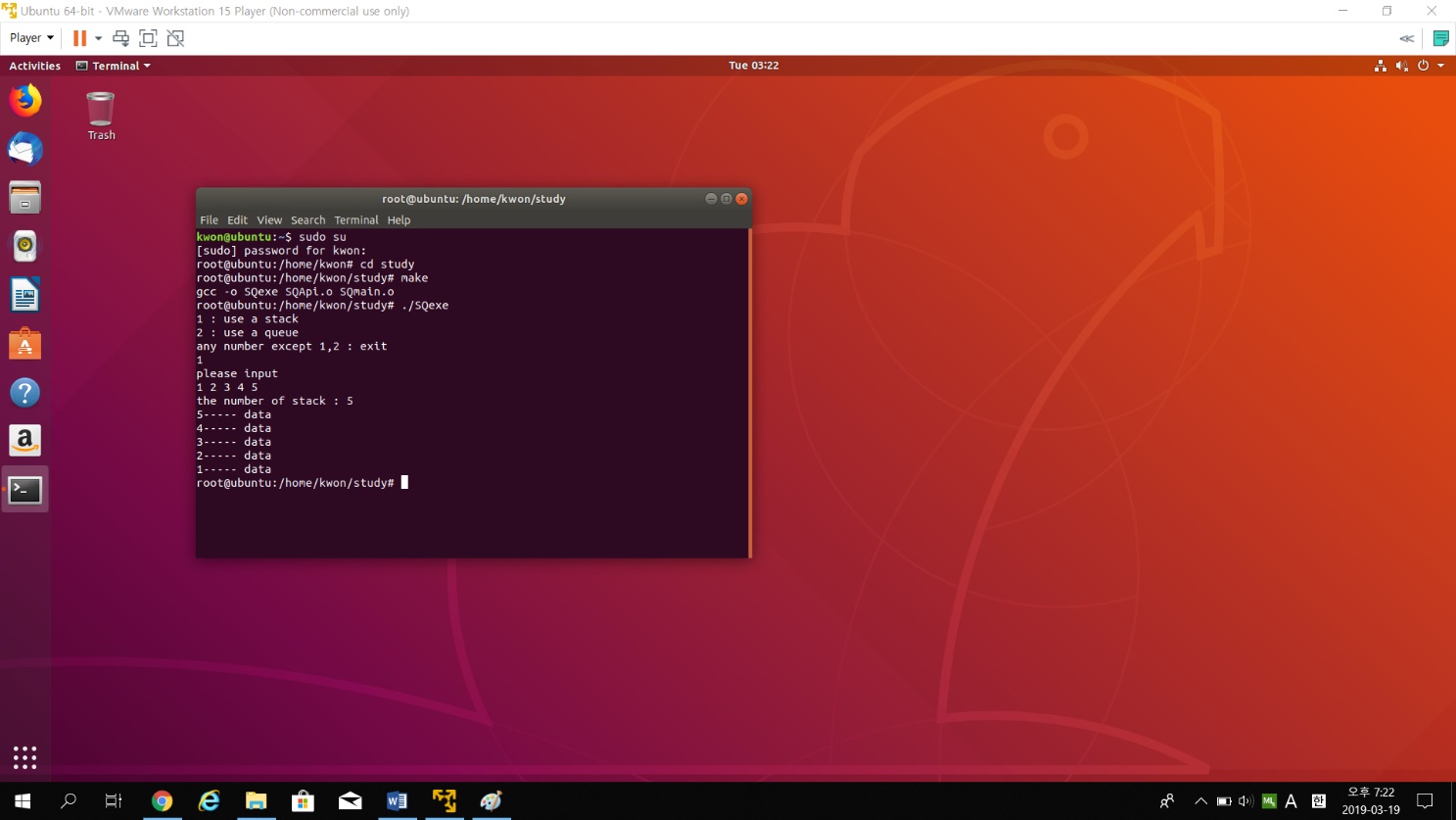
인식 후에 stack이나 queue에 있는 정보를 모두다 out한다.

2.2 다이어그램

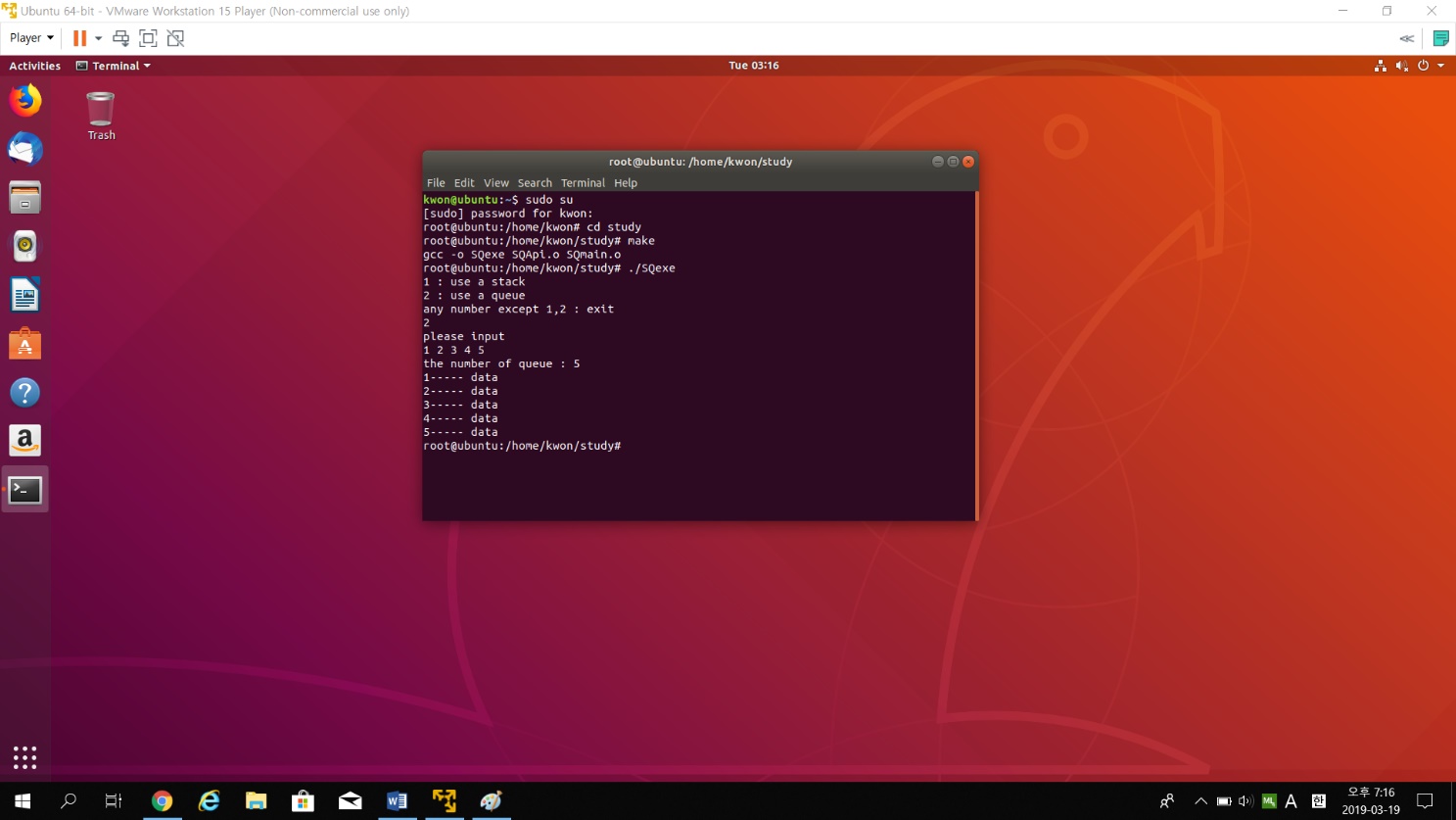


3. 실행 결과

Stack 결과



Queue 결과



**(실행결과는 자신의 컴퓨터환경에서 실행한 결과 캡쳐(그림파일)해서 문서에 첨부하여 제출합니다. 실행결과가 성공한 경우뿐만 아니라 실패한 경우도 포함되어야 함.**

**만약 과제로 올려서 실행한 결과와 실행 결과로 캡쳐한 결과가 다를 경우 감점합니다.)**

4. 고찰 (제일 중요함 - **점수배점이 가장 높음!** 자세히 쓸 것)

이번 프로그래밍에서 가장 힘들었던 것은 리눅스 환경에 적응하는 것이었다.

Queue와 Stack은 지난 학기 자료구조 때 해봤기 때문에 어렵지는 않았지만 리눅스 환경에서

Terminal로 하려니 골치가 아팠다. 명령어도 외워야 했고 무엇보다 윈도우 환경에서 쓰던

단축키들이 먹히질 않으니 코딩하는데 너무 불편했다. 복사 붙여넣기도 굉장히 불편했고, 특히

평소 윈도우에서 코딩하다 습관적으로 쓰던 ctrl+s(저장) 키를 한번 누른 적이 있었는데, 갑자기

먹통이 되어서 하던 작업을 날리곤 했다. 결국 나중에는 그것이 터미널의 문자 출력을 중단하는

키이고 ctrl+q를 누르면 해결 할 수 있다는 것을 알게 되었지만, 꽤 골머리를 앓았던 문제였

다. 하지만 스스로 인터넷 검색이나 교수님 ppt를 참고하며 리눅스와 씨름하다보니 조금

적응이 된듯하다. 앞으로 좀 더 리눅스와 친해져야 앞으로의 코딩이 편할 것이라 생각된다.

과제가 없어도 리눅스로 코딩을 하는 연습을 계속해봐야겠다.

5. 프로그램 소스 파일 (주석을 꼭 달 것)

-SQ.h

typedef struct stack{

void\* data;

struct stack\* next;

}Stack;

typedef struct stackHead{

int dataNumber; // the number of data

Stack\* head; // always pointing a head of stack

}S\_Head;

S\_Head\* MakeStack();// make a Stack head

int PushStack(S\_Head\* stack,void\* data); // put in stack

void\* PopStack(S\_Head\* stack); //out from stack

typedef struct queue{

void\* data;

struct queue\* next;

}Queue;

typedef struct queueHead{

int dataNumber;

Queue\* head;

Queue\* rear;

}Q\_Head;

Q\_Head\* MakeQueue();// make a Queue head

int Enqueue(Q\_Head\* head,void\* data); //put in queue

void\* Dequeue(Q\_Head\* head); // out from queue

-SQApi.c

#include "SQ.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

S\_Head\* MakeStack(){

S\_Head\* stack = (S\_Head\*)malloc(sizeof(S\_Head));

if(stack == NULL){

return NULL;

}

stack->dataNumber = 0;

stack->head = NULL;

return stack;

}

int PushStack(S\_Head\* stack,void\* data){

if(stack != NULL){//not strange input value

char\* newData = (char\*)malloc(sizeof(char)\*10);

strcpy(newData,data);

Stack\* temp = (Stack\*)malloc(sizeof(Stack));

if(temp == NULL){

return 0; // failed

}

temp->data = newData;

temp->next = NULL;

if(stack->dataNumber == 0){

stack->head = temp;

}else{

temp->next = stack->head;

stack->head = temp;

}

stack->dataNumber++;

return 1; // success

}

return 0; //failed

}

void\* PopStack(S\_Head\* stack){

if(stack->dataNumber != 0){

Stack\* temp = stack->head;// temporary save in temp

//

stack->head = stack->head->next;

stack->dataNumber--;

//

void\* pop = temp->data;

free(temp);

return pop;

}else{

printf("no more data in stack");

return NULL;

}

}

Q\_Head\* MakeQueue(){

Q\_Head\* head = (Q\_Head\*)malloc(sizeof(Q\_Head));

if(head == NULL){

return NULL;

}

head->dataNumber = 0;

head->head = NULL;

head->rear = NULL;

return head;

}

int Enqueue(Q\_Head\* head, void\* data){

//Queue initialize

Queue\* queue = (Queue\*)malloc(sizeof(Queue));

if(queue == NULL){

return 0;//failed

}

char\* newData = (char\*)malloc(sizeof(char)\*10);

strcpy(newData,data);

queue->data = newData;

queue->next = NULL;

if(head->dataNumber == 0){

//Enqueue

head->head = queue;

head->rear = queue;

}else{// >=1

head->rear->next = queue;

head->rear = queue;

}

head->dataNumber++;

return 1;

}

void\* Dequeue(Q\_Head\* head){

if(head->dataNumber == 0){

printf("no data in Queue");

return NULL;

}

void\* temp = head->head->data;

if(head->dataNumber==1){

free(head->head);

head->head = NULL;

head->rear = NULL;

head->dataNumber--;

}else{

Queue\* queue = head->head;

head->head = head->head->next;

free(queue);

head->dataNumber--;

}

return temp; // success

}

SQmain.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "SQ.h"

void SQ\_InputFunction(void\* SQ,int type){ // function for input

// 0: stack input 1: queue input

char input[20];

char\* temp;

printf("please input\n");

fgets(input,sizeof(input),stdin);

temp = strtok(input," \n");

while(temp != NULL){

if(type == 0){

PushStack(SQ,temp);

}else if(type == 1){

Enqueue(SQ,temp);

}

temp = strtok(NULL," \n");

}

}

void SQ\_OutputFunction(void\* SQ,int type){ // function for output

if(type == 0){ // in case stack

while(((S\_Head\*)SQ)->dataNumber !=0){

void\* temp = PopStack(SQ);

printf((char\*)temp);

free(temp);

printf("----- data\n");

}

}else if(type == 1){

while(((Q\_Head\*)SQ)->dataNumber !=0){

void\* temp = Dequeue(SQ);

printf((char\*)temp);

free(temp);

printf("----- data\n");

}

}

}

int main(){

printf("1 : use a stack\n2 : use a queue\nany number except 1,2 : exit \n");

char ch = getchar();getchar();

if(ch == '1'){ // stack

S\_Head\* stack = MakeStack();

SQ\_InputFunction(stack,0);

printf("the number of stack : %d \n",stack->dataNumber);

SQ\_OutputFunction(stack,0);

}else if(ch == '2'){ // queue

Q\_Head\* queue = MakeQueue();

SQ\_InputFunction(queue,1);

printf("the number of queue : %d \n",queue->dataNumber);

SQ\_OutputFunction(queue,1);

}

}

6. 자료 출처 (어떤 책을 참고했으며 어떤 웹사이트를 참고했는지 출처를 적습니다)

대부분 교수님의 ppt를 참고했습니다.