1, 개요

리눅스 환경에서 Stack, queue를 직접 짜고 실행해보며 Strtok 함수를 이용하여 자료구조(특히 stack, queue)에 대한 이해도를 높이고, Makefile의 사용법을 익힙니다.

2. 프로그램 구조 설명

2.1 함수에 대한 설명

STACK\* create\_stack () : 스택을 만드는 함수입니다.

bool push(STACK\* stack, void\* data) : 스택에 보이드형 데이터를 주소형으로 넣는 함수입니다.

void\* pop(STACK\* stack) : 스택의 맨 위 데이터를 보이드 주소형으로 꺼내는 함수입니다.

QUEUE\* create\_queue() : 큐를 만드는 함수입니다.

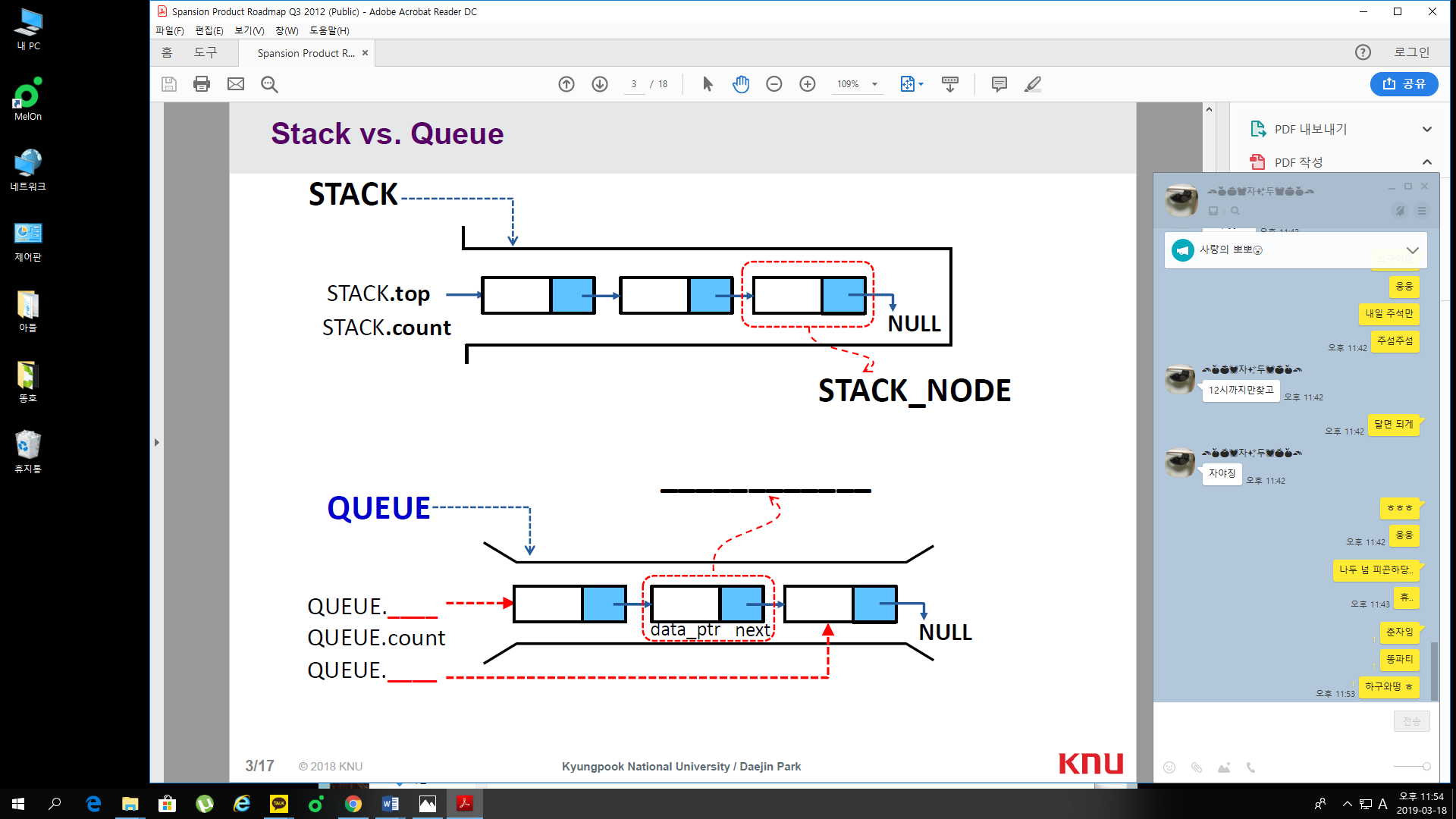
bool enqueue(QUEUE\* queue, void\* input) : 큐에 보이드형 데이터를 주소형으로 넣는 함수입니다

void\* dequeue(QUEUE\* queue) : 큐의 가장 앞부분에 있는 자료를 보이드 주소형으로 꺼내는 함수입니다.

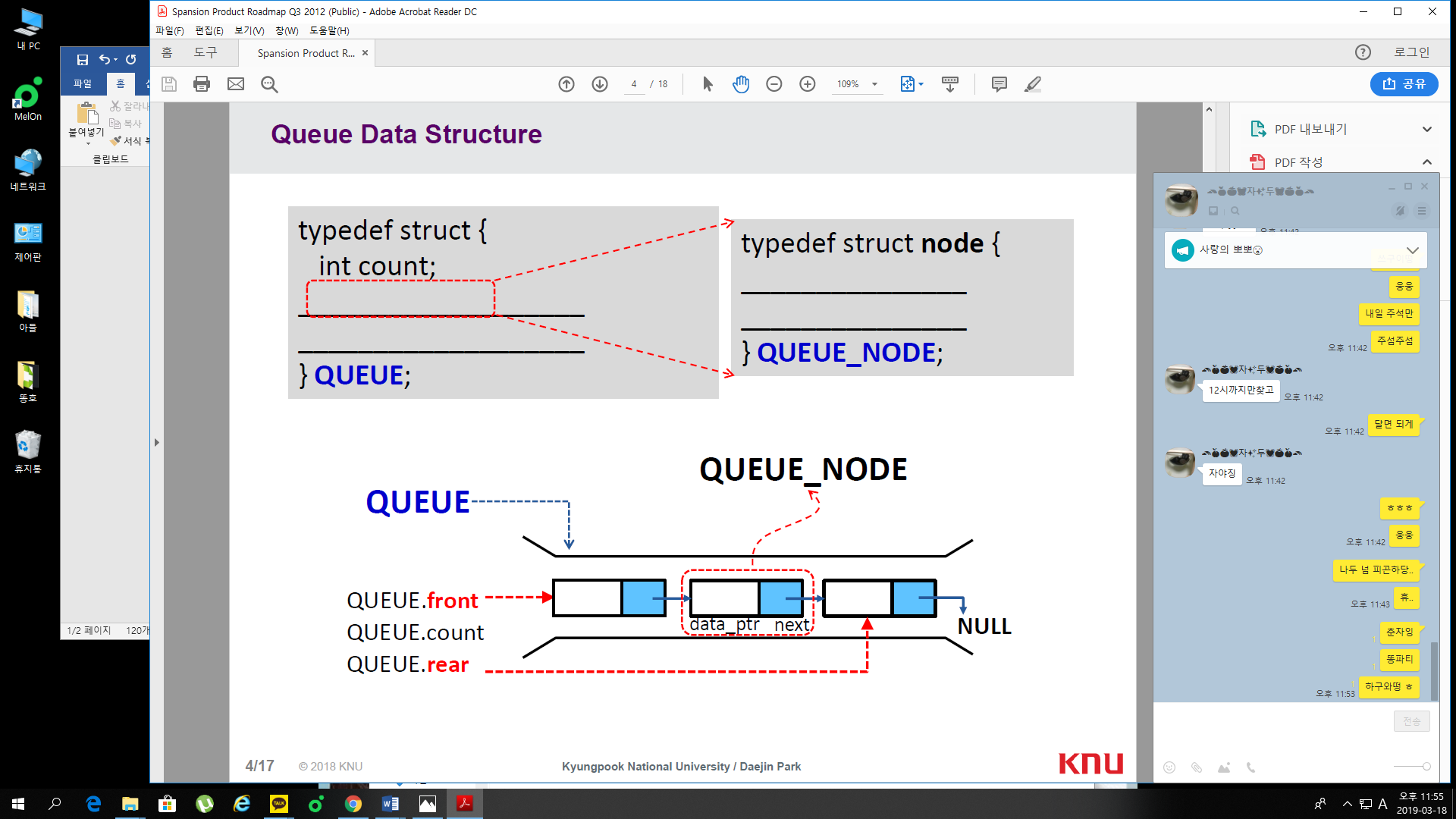
gets(문자열, 문자열의 크기) : 문자열을 입력받는 함수입니다.

strtok(문자열, “구분자”) : 문자열을 구분자가 있는곳마다 자르는 함수입니다.

2.2 다이어그램

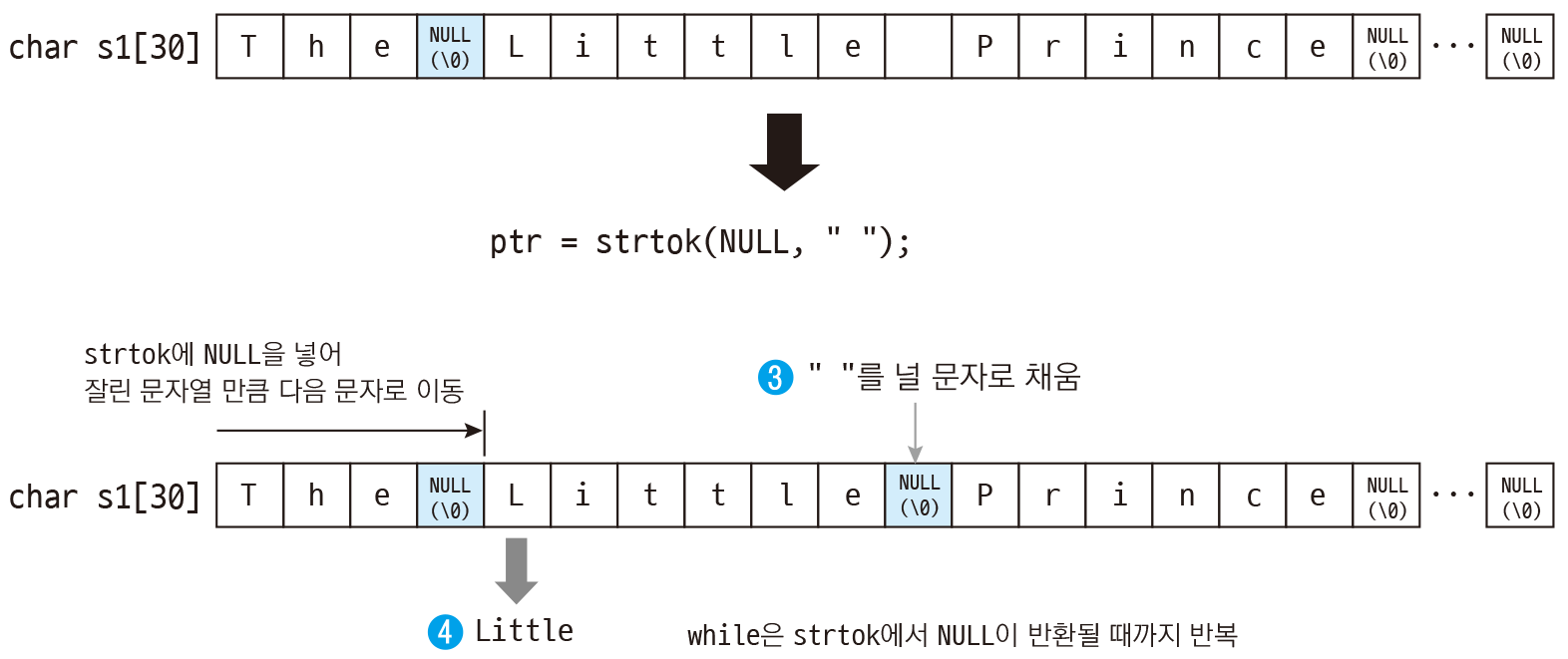
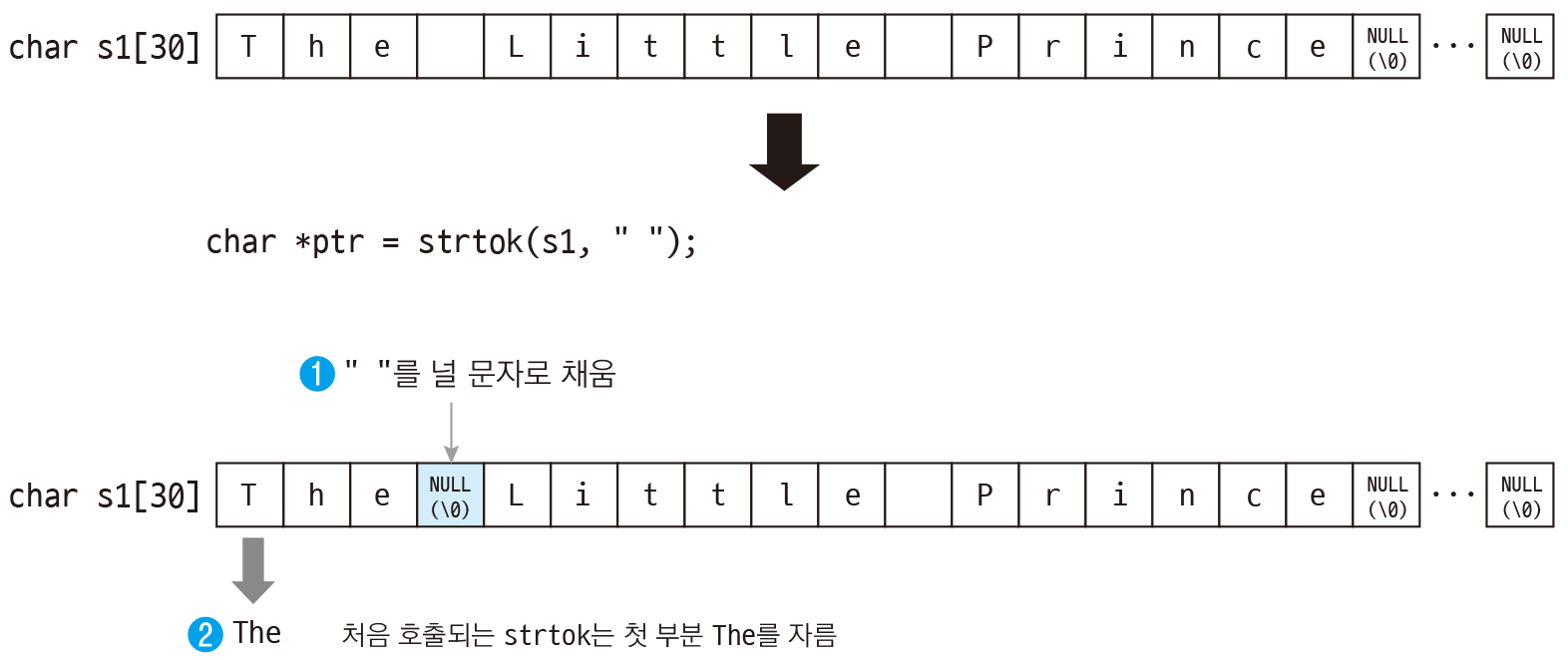


스택의 다이어그램입니다. push함수를 이용하여 스택에 자료를 추가하고, pop함수를 이용하여 가장 최근에 들어간 자료를 추출합니다. Ex) 1 2 3 4 5 -> 5 4 3 2 1

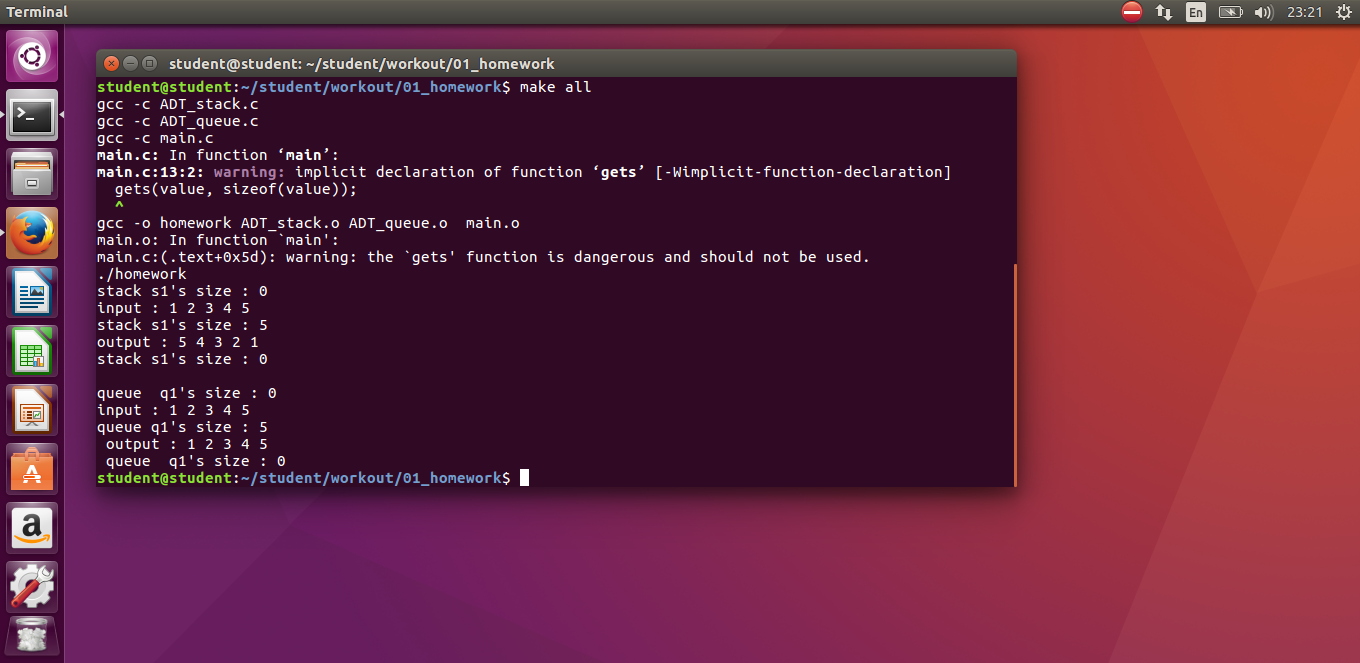


큐의 다이어그램입니다. enqueue함수를 이용하여 큐에 자료를 추가하고, dequeue함수를 이용하여 가장 마지막에 들어간 자료를 추출합니다. Ex) 1 2 3 4 5 -> 1 2 3 4 5

다음은 strtok함수의 실행원리입니다.



3. 실행 결과



4. 고찰

보이드포인터와 자료의 동적할당은 정말 유용하지만 사용할 때 항상 신중해야한다는 점에서 날카로운 칼을 다루는것 같다는 생각이 들었습니다. 자료구조에서 배운 stack과 queue을 리눅스 환경에서 시행해보며 단축키들을 익히고 makefile을 만들며 힘든 부분이 있었지만 이제는 좀 알 것 같습니다.

5. 프로그램 소스 파일

typedef struct node{

void\* data\_ptr;

struct node\* link;

}STACK\_NODE; //스택에 사용될 노드의 구조체입니다.

typedef struct{

int count;

STACK\_NODE\* top;

}STACK; //스택의 구조체 입니다.

STACK\* create\_stack(){ // 스택을 만드는 create\_stack 함수입니다

STACK\* stack; //스택 구조체를 만듭니다.

stack = (STACK\*)malloc(sizeof(STACK)); // 스택에 메모리 동적할당을 해줍니다.

stack -> top = NULL;

stack -> count = 0;

return stack; // 만들어낸 스택을 return합니다.

}

bool push(STACK\* stack, void\* data){ // 스택에 데이터를 넣는 push 함수입니다.

STACK\_NODE\* add\_stack; // 데이터를 넣을 add\_stack노드를 만듭니다.

add\_stack = (STACK\_NODE\*)malloc(sizeof(STACK\_NODE)); // add\_stack노드를 동적할당합니다

add\_stack -> data\_ptr = data; // add\_stack노드에 넣고자하는 데이터를 넣습니다.

add\_stack -> link = stack->top; // add\_stack노드를 스택의 top과 연결합니다.

stack->top = add\_stack; //스택의 top을 add\_stack노드로 지정합니다.

(stack -> count++); // stack의 count를 늘려줍니다.

return true;

}

void\* pop(STACK\* stack){ // 스텍에서 데이터를 추출하는 pop함수입니다.

if(stack->count ==0) return false; // 스택의 자료가 없을 때 예외처리입니다.

STACK\_NODE\* pop\_stack; // 뽑아낼 노드를 지정할 pop\_stack 포인터를 만들어줍니다.

pop\_stack = stack->top; // pop\_stack으로 스택의 top을 지정합니다

void\* value = stack->top->data\_ptr; // 데이터를 보이드 포인터 값으로 추출합니다.

stack -> top = pop\_stack -> link; // top을 재설정합니다.

pop\_stack -> link = NULL; // pop\_stack의 링크를 삭제하여 free해도 문제가 없게 합니다.

free(pop\_stack); // 동적할당으로 만들어낸 자료이므로 free합니다.

pop\_stack = NULL; // pop\_satck 포인터도 null값을 넣습니다.

(stack -> count)--; // 스택의 count도 줄여줍니다.

return value; // 추출해낸 데이터를 return합니다.

}

typedef struct node\_Q{

void\* data\_ptr;

struct node\_Q\* next;

}QUEUE\_NODE; //큐에 사용될 노드의 구조체입니다.

typedef struct {

int count;

QUEUE\_NODE\* front;

QUEUE\_NODE\* rear;

}QUEUE; //큐의 구조체입니다.

QUEUE\* create\_queue(){ //큐를 만드는 함수입니다.

QUEUE\* new\_queue; // 새로운 큐를 만듭니다

new\_queue =(QUEUE\*)malloc(sizeof(QUEUE)); // 메모리 동적할당을 해줍니다.

if(new\_queue){ // 큐가 만들어 진 상태라면 프론트, 리어 주소값에 널값을 주고 카운트 0를 부여합니다.

new\_queue -> front = NULL;

new\_queue -> rear = NULL;

new\_queue -> count = 0;

return new\_queue; // 새로 만들어진 큐 값을 return합니다.

} else{

return NULL; // 예외처리

}

}

bool enqueue(QUEUE\*queue, void\* input){ // 큐에 데이터를 넣는 enqueue함수입니다.

QUEUE\_NODE\* fresh\_node; // 큐에 들어갈 데이터를 담는 fresh\_node를 만듭니다

fresh\_node = (QUEUE\_NODE\*)malloc(sizeof(QUEUE\_NODE)); // fresh\_node를 메모리 동적할당 해줍니다.

fresh\_node -> data\_ptr = input; // 데이터를 fresh\_node 의 data\_ptr에 넣습니다.

fresh\_node -> next = NULL; // fresh\_node는 rear에 들어갈 것이므로 next는 null입니다.

if(queue->count == 0){ // 큐가 텅 빈 경우입니다.

queue -> front = fresh\_node; // 큐의 front와 rear 모두 새로 만들어진 노드입니다.

queue -> rear = fresh\_node;

(queue->count)++;

} else{ // 큐에 데이터가 하나라도 있는 경우입니다.

queue-> rear -> next = fresh\_node; // 큐의 rear를 재설정해줍니다.

queue -> rear = fresh\_node; // 바로앞의 코드와 순서가 헷갈리면 안됩니다.

(queue->count)++;

}

return true;

}

void\* dequeue(QUEUE\* queue){ //큐에서 데이터를 빼내는 dequeue함수입니다.

if(queue -> count ==0) false; //큐에 데이터가 하나도 없을 때 예외처리입니다.

QUEUE\_NODE\* byebye\_node; //사라질 큐의 노드를 지정할 byebye\_node 포인터를 만듭니다.

byebye\_node = queue -> front; // 큐의 가장 앞부분을 byebye\_node포인터로 지정합니다.

void\* output = byebye\_node -> data\_ptr; // 데이터를 추출합니다.

if(queue -> count ==1) { // 큐의 데이터가 하나일 케이스입니다.

queue->front = NULL;

queue->rear = NULL;

}else{ //queue의 데이터가 하나도, 0개도 아닐 케이스입니다.

queue -> front = queue -> front ->next;

}

free(byebye\_node); // malloc으로 동적할당 했으니 free를 해줍니다.

queue->count --; //큐의 개수를 빼줍니다.

return output; // 추출한 데이터를 return합니다

}

int main(){

STACK\* s1 = create\_stack(); // 스택 s1을 만듭니다.

printf("stack s1's size : %d\n", s1 ->count); //우선 스택에 데이터를 넣기 전 자료의 수를 말해줍니다.

printf("input : ");

char value[10];

gets(value, sizeof(value)); //gets함수를 이용해 문자열을 입력받습니다.

char\* tok;

tok = strtok(value," "); // strtok함수를 이용해 문자열에 공백이 있을 때 끊습니다.

while (tok!=NULL){ // 반복문을 사용하여 계속 끊어줍니다

push(s1,tok); // push함수를 이용해 스택 s1에 strtok을 이용해 자른 데이터를 넣습니다.

tok = strtok(NULL, " ");

}

printf("stack s1's size : %d\noutput : ", s1 ->count); // 스택에 데이터를 넣고 난 이후 자료의 수를 말해줍니다.

char\* temp;

while (s1->count != 0){ // 반복문과 pop 함수를 이용하여 스택의 자료를 모두 추출합니다

temp = (char\*)pop(s1);

printf("%c ",\*temp);

}

printf("\nstack s1's size : %d\n\n", s1 ->count); // 스택의 데이터를 추출하고 난 다음의 자료 수를 말해줍니다.

QUEUE\* q1 = create\_queue(); // 큐 q1을 만듭니다.

printf("queue q1's size : %d\n", q1 ->count); // 큐에 데이터를 넣기 전 자료의 수를 말해줍니다.

printf("input : ");

gets(value, sizeof(value)); //gets함수를 이용해 문자열을 입력받습니다.

tok = strtok(value," "); //strtok 함수를 이용해 문자열에 공백이 있을 때 끊습니다.

while (tok!=NULL){ //반복문을 사용하여 계속 끊어줍니다.

enqueue(q1,tok); //enqueue 함수를 이용해 큐 q1에 strtok을 이용해 자른 데이터를 넣습니다.

tok = strtok(NULL, " ");

}

printf("queue q1's size : %d\n output : ", q1 ->count); // 큐에 데이터를 넣고 난 다음 자료의 수를 말해줍니다.

while (q1->count != 0){ //반복문과 dequeue 함수를 이용하여 큐의 자료를 모두 추출합니다.

temp = (char\*)dequeue(q1);

printf("%c ",\*temp);

}

printf("\n queue q1's size : %d\n", q1 ->count); // 큐의 데이터를 추출하고 난 다음의 자료 수를 말해줍니다.

return 0;

}

6. 자료 출처

자료구조 – 박대진교수님 자료구조 수업자료

strtok 함수 - https://dojang.io/mod/page/view.php?id=376