

REPORT

과 목	데이터 구조 (7-9)
담당 교수	김인겸 교수님
학 과	정보통신공학과
학 번	20190895
이 름	김찬영
제 출 일	2022.06.15

목차

1. 과제 목표 분석

2. 프로그램 코딩

3. 프로그램 실행 결과

1. 과제 목표 분석

Linked List 를 이용한 큐와 스택을 사용하여 스케줄러를 작성하시오.

(스케줄러는 단순하게 할 일만을 입력 받아서 main() 함수의 argument 에 따라 입력한 순서대로 혹은 입력한 역순으로 알려주는 역할을 하는 것으로 구성한다)

Node 구성은 아래와 같이 정의하여 사용한다. 최대 입력은 256 글자 이내로 구성한다.

```
typedef char Element;
typedef struct ListNode {
    Element schedule[256];
    struct ListNode* link;
} Node;
```

Main()함수에서는 아래와 같이 command 에 따라 스케줄 입력과 display 를 구동하는 작업을 구현한다.

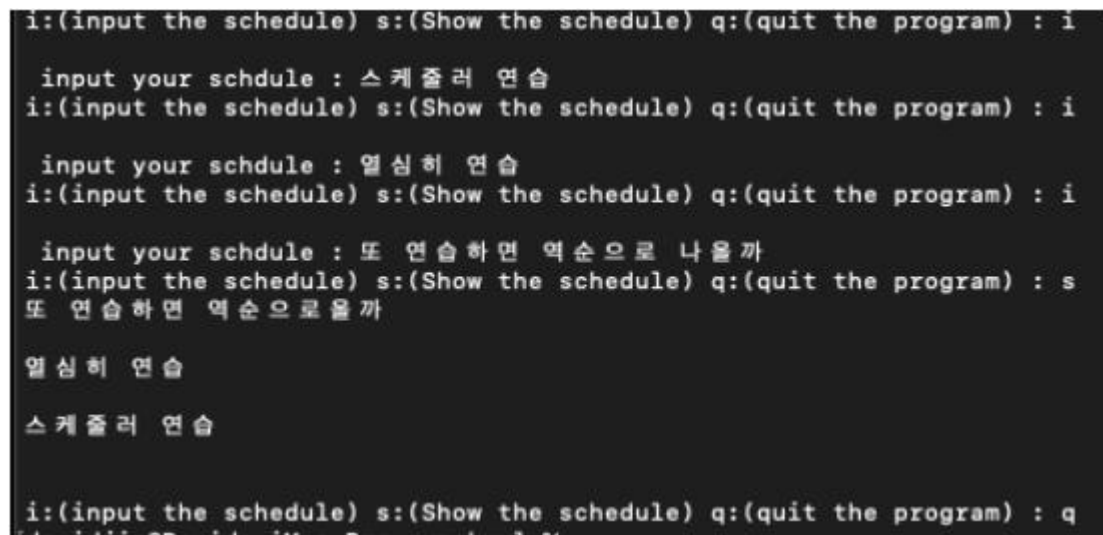
```
char command;
Node Schedule;
```

```
init_stack();
```

```
printf("I:(input the schedule) s:(Show the schedule) q:(quit the program) : ");
```

입력은 fgets(Schedule.schedule, 256, stdin); 으로 받는다.

아래의 예는 스택으로 구현한 결과를 보여주는 스케줄 프로그램의 예이다. 큐와 스택에 대하여 모두 동작하는 프로그램을 완성한 후 리포트로 제출합니다.



```
i:(input the schedule) s:(Show the schedule) q:(quit the program) : i
    input your schdule : 스케줄러 연습
i:(input the schedule) s:(Show the schedule) q:(quit the program) : i
    input your schdule : 열심히 연습
i:(input the schedule) s:(Show the schedule) q:(quit the program) : i
    input your schdule : 또 연습하면 역순으로 나올까
i:(input the schedule) s:(Show the schedule) q:(quit the program) : s
    또 연습하면 역순으로 올까

열심히 연습

스케줄러 연습

i:(input the schedule) s:(Show the schedule) q:(quit the program) : q
```

우선 문제 해결에 앞서 필요한 것들에 대해서 생각해 보았다.

(1) 연결 리스트로 스택과 큐의 구현

교재에 나와있는 연결리스트로 구현한 스택(p.179), 연결리스트로 구현한 큐(p.186)를 참고하여 구현할 수 있다.

(2) Node 에는 문자형 배열을 저장

먼저 정의한 Element 를 char 형으로 바꿔주면 간단히 구현할 수 있다. 또한, 큐와 스택 안에 문자열을 저장하기 때문에, 삽입 연산과 삭제 연산도 문자열을 삽입하거나 문자열을 반환해주는 형식으로 구현해야 한다.

(3) Check 가 0 일 때는 큐를 사용해 동작, 1 일 때는 스택을 사용해 동작

`main(int argc, char* argv[])`를 사용하여 `main()`함수의 argument 로 check 값을 입력받은 후, `argv[1]`의 값을 `atoi()` 함수를 사용해 `int check` 에게 전달하여 구분할 수 있다.

(4) 입력받은 스케줄을 출력하는 `display()` 함수 정의

스케줄을 입력받은 뒤 큐 또는 스택에 저장하게 될 텐데, 조건문 안에 `is_empty()`함수를 사용하면 출력을 할 때, 큐와 스택을 모두 비울 때 까지 `dequeue()` 또는 `pop()`을 해서 `printf()`함수를 사용해 출력할 수 있다.

(5) command 값에 따라 스케줄을 입력받거나 출력

교재에 나와있는 program6.17 리스트를 이용한 라인 편집기 프로그램 (p.234)을 참고하여 구현하였다. Command = I 일 때는 입력 모드인데, 입력 모드에서는 스케줄을 입력하라는 문구를 출력한 뒤, `fgets(Schedule.schedule, 256, stdin)`함수를 사용해 입력을 받고 큐일 경우 `enqueue()`, 스택일 경우 `push()`를 실행하면 입력이 완료된다.

Command = s 일 때는 출력 모드인데, 출력 모드의 경우 앞서 정의한 `display()` 함수를 불러와 모든 스케줄을 출력하면 출력이 완료된다. 또한 command = q 일 때는 프로그램 종료인데, 단순히 반복문을 탈출하는 것으로 종료시킬 수 있다.

이것들을 do-while 문으로 구현하면 사용자가 q 를 입력하기 전 까지 계속해서 스케줄의 입력 및 출력을 사용할 수 있다.

2. 프로그램 코딩

(1) 헤더 파일 포함, 에러 출력 함수, my_fflush(), 연결리스트 이용한 스택의 구현

//헤더 파일은 다음과 같이 포함시켰다

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> //strcpy() 함수 사용
```

//에러 출력 함수

```
void error(char* str){
    fprintf(stderr, "%s\n", str);
    exit(1); }
```

//입력 버퍼(키보드)에서 엔터키를 입력했을 시의 개행문자를 제거. 교재 program 6.17에 포함

```
void my_fflush() {
    while (getchar() != '\n'); }
```

//연결리스트 이용한 스택

```
typedef char Element;
typedef struct LinkedNode {
    Element schedule[256];
    struct LinkedNode* link;
} Node;
Node* top = NULL;
```

//스택이 비어있는지 확인하는 함수

```
int is_empty() { return top == NULL; }
```

//스택 초기화 함수

```
void init_stack() { top = NULL; }
```

//스택 안의 요소의 개수를 반환하는 함수

```
int size() {
    Node* p;
    int count = 0;
    for (p = top; p != NULL; p = p->link) count++;
    return count;
}
```

//스택 삽입 연산 : 문자열을 노드에 저장해야 하기 때문에 문자열 전달받음

```
void push(Element e[]) {
    Node* p = (Node*)malloc(sizeof(Node));
```

//노드의 data에 문자열을 저장해야 하기 때문에 strcpy() 함수 사용

```
strcpy(p->schedule, e);
p->link = top;
top = p;
```

```
}
```

//스택 pop연산

```
Element* pop() {  
    Node* p;  
  
    //문자열을 반환하기 위해 문자열 선언  
    Element e[256];  
  
    if (is_empty())  
        error("스택 공백 에러");  
    p = top;  
    top = p->link;  
  
    //pop()하는 노드의 문자열을 e로 복사  
    strcpy(e, p->schedule);  
    free(p);  
    return e;  
}
```

//스택 peek연산

```
Element* peek() {  
    if (is_empty())  
        error("스택 공백 에러");  
    return top->schedule;  
}
```

//스택 화면 출력 함수 : 스택이 비어있을 때 까지 pop() 출력

```
void display() {  
    while (!is_empty()) {  
        printf("%s\n", pop());  
    }  
}
```

(2) 연결리스트를 이용한 큐의 구현

```
typedef char Element_queue;
typedef struct LinkedNode_queue {
    Element_queue schedule_queue[256];
    struct LinkedNode_queue* link_queue;
} Node_queue;

//front와 rear의 선언
Node_queue* front = NULL, * rear = NULL;

//큐가 비어있는지 확인하는 함수
int is_empty_queue() { return front == NULL; }

//큐를 초기화하는 함수
void init_queue() { front = rear = NULL; }

//큐 안의 요소의 개수를 반환하는 함수
int size_queue() {
    Node_queue* p;
    int count = 0;
    for (p = front; p != NULL; p = p->link_queue) count++;
    return count;
}

//큐 삽입 연산 : 문자열을 노드에 저장해야 하기 때문에 문자열을 전달받음
void enqueue(Element_queue e[]) {
    Node_queue* p = (Node_queue*)malloc(sizeof(Node_queue));

    //노드의 data에 문자열을 저장해야 하기 때문에 strcpy()함수 사용
    strcpy(p->schedule_queue, e);
    p->link_queue = NULL;
    if (is_empty_queue()) front = rear = p;
    else {
        rear->link_queue = p;
        rear = p;
    }
}

//큐 삭제 연산
Element_queue* dequeue() {
    Node_queue* p;

    //문자열을 반환하기 위해 문자열로 선언
    Element_queue e[256];

    if (is_empty_queue())
        error("큐 공백 오류");
    p = front;
    front = front->link_queue;
    if (front == NULL) rear = NULL;

    //dequeue하려는 노드의 문자열을 e로 복사
    strcpy(e, p->schedule_queue);
    free(p);
    return e;}
```

//큐 peek 연산

```
Element_queue* peek_queue() {  
    if (is_empty_queue())  
        error("큐 공백 오류");  
    return front->schedule_queue;  
}
```

//큐 화면 출력 함수 : 큐가 비어있을 때 까지 dequeue() 출력

```
void display_queue() {  
    while (!is_empty_queue()) {  
        printf("%s\n", dequeue());  
    }  
}
```


(3) main()함수

```
int main(int argc, char* argv[]) { //argument로 입력받음

    int check;
    char command;
    Node Schedule;
    Node_queue Schedule_queue;

    //올바르지 않은 명령어 입력 시 에러 출력
    if (argc != 2) {
        error("exe check(0 or 1)\n");
    }

    //0번 큐 모드로 실행할 것인지, 1번 스택 모드로 실행할 것인지 결정하기 위해 사용
    check = atoi(argv[1]);

    //스택 및 큐의 초기화
    init_stack();
    init_queue();

    //check의 값이 0일 때 큐를 이용하여 실행
    if (check == 0) {
        printf("\n큐를 이용하여 스케줄러를 실행합니다. 입력한 순서대로 출력됩니다.\n");
        do {

            printf(" I:(input the schedule) s:(Show the schedule) q:(quit the program) : ");
            command = getchar();
            my_fflush();
            switch (command) {
                case 'i': //스케줄 입력 모드
                    printf(" Input your schedule : ");
                    fgets(Schedule_queue.schedule_queue, 256, stdin);
                    //node_queue의 멤버인 schedule_queue에 스케줄 내용을 입력받음
                    enqueue(Schedule_queue.schedule_queue); //입력받은 스케줄을 큐에 삽입
                    break;

                case 's': //스케줄 출력 모드 : 큐는 선입선출 방식이기에 입력한 순서대로 출력함
                    display_queue();
                    break;

                case 'q': //스케줄러 종료
                    printf("\n프로그램을 종료합니다.\n");
                    break;
            }
        } while (command != 'q');
    }
}
```

```
//check의 값이 1일 때 스택을 이용하여 실행
```

```
else if (check == 1) {  
    printf("\n스택을 이용하여 스케줄러를 실행합니다. 입력한 역순으로 출력됩니다.\n");  
    do {  
  
        printf("I:(input the schedule) s:(Show the schedule) q:(quit the program) : ");  
        command = getchar();  
        my_fflush();  
        switch (command) {  
            case 'i'://스케줄 입력 모드  
                printf(" Input your schedule : ");  
                fgets(Schedule.schedule, 256, stdin);  
                //node의 멤버인 schedule에 스케줄 내용을 입력받음  
                push(Schedule.schedule);//입력받은 스케줄을 스택에 삽입  
                break;  
  
            case 's'://스케줄 출력 모드 : 스택은 후입선출 방식이기에 입력한 순서의 반대로  
                출력함  
                display();  
                break;  
  
            case 'q'://스케줄러 종료  
                printf("\n프로그램을 종료합니다.\n");  
                break;  
        }  
    } while (command != 'q');  
}  
  
//check값이 0 또는 1이 아닐 경우 에러 출력  
else {  
    error("exe check(0 or 1)\n");  
}  
  
return 0;  
}
```

3. 프로그램 실행 결과

(1) 큐를 이용한 스케줄러 실행

아래는 < 파일이름.exe 0 > 명령어를 입력하여 큐를 이용한 스케줄러를 실행하고 직접 스케줄을 추가한 뒤, 출력하는 프로그램의 동작을 나타낸다. 큐를 사용하여 스케줄을 저장하였기 때문에, 처음 입력한 문장의 순서대로 출력되는 것을 확인할 수 있다.

```
C:\Users\User>C:\Users\User\source\repos\데이터구조기말고사\64\Debug\데이터구조기말고사.exe 0
큐를 이용하여 스케줄러를 실행합니다. 입력한 순서대로 출력됩니다.
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 안녕하세요
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 사랑합니다
l:(input the schedule) s:(Show the schedule) q:(quit the program) : s
안녕하세요
사랑합니다
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 첫번째 문장입니다.
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 두번째 문장입니다.
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 세번째 문장입니다.
l:(input the schedule) s:(Show the schedule) q:(quit the program) : s
첫번째 문장입니다.
두번째 문장입니다.
세번째 문장입니다.
l:(input the schedule) s:(Show the schedule) q:(quit the program) : q
프로그램을 종료합니다.
```

(2) 스택을 이용한 스케줄러 실행

아래는 < 파일이름.exe 1 > 명령어를 입력하여 스택을 이용한 스케줄러를 실행하고 직접 스케줄을 추가한 뒤, 출력하는 프로그램의 동작을 나타낸다. 스택을 사용하여 스케줄을 저장하였기 때문에, 처음 입력한 문장의 역순으로 출력되는 것을 확인할 수 있다.

```
C:\Users\User>C:\Users\User\source\repos\데이터구조기말고사\64\Debug\데이터구조기말고사.exe 1
스택을 이용하여 스케줄러를 실행합니다. 입력한 역순으로 출력됩니다.
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 첫 번째 문장을 입력
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 두 번째 문장을 입력
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 세 번째 문장을 입력
l:(input the schedule) s:(Show the schedule) q:(quit the program) : i
Input your schedule : 마지막 문장을 입력
l:(input the schedule) s:(Show the schedule) q:(quit the program) : s
마지막 문장을 입력
세 번째 문장을 입력
두 번째 문장을 입력
첫 번째 문장을 입력
l:(input the schedule) s:(Show the schedule) q:(quit the program) : q
프로그램을 종료합니다.
```