자료구조 (DS)

<과제 1>
제공된 동영상과 프로그램 파일을 참고하여 연결리스트를 이용하여 구현한 스택 프로그램을 만들 수 있는 능력을 키우세요.

연결리스트를 이용하여 구현한 스택 프로그램 제작 및 제작 능력 개발

한라대학교 ICT 공학융합부
정보통신소프트웨어학과

201932030

송현교

(첨부된 사진은 구현 모습만 나타내고 있습니다. 대신 코드를 복사하실 수 있게끔 작성해두었습니다.)
(코드의 가독성을 위해 간소화하지 않고 그대로 작성하여 한 페이지에 코드를 모두 담을 수 없었던 점 죄송합니다.)

목차

1. 연결 리스트로 구조체 생성

2. IsEmpty() 를 생성, 이용하여 스택의 공백 여부를 확인해보자.

3. push(data e)를 이용하여 창고에 값을 집어넣어 보자.

4. pop() 을 생성, 이용하여 창고의 맨 위에 있는 물건을 꺼내보자.

5. peek()를 생성, 이용하여 창고의 맨 위에 있는 물건이 무엇인지 알아보자.

6. size()로 배열의 크기 알아내기, clear() 로 배열을 초기화 시키고 프로그램을 완성시키자.

1) 연결 리스트로 구조체 생성

구현 및 구현 모습 :

```
 #define _CRT_NO_SECURE_WARNINGS
#include <stdio.h>
 #include <malloc.h> //

struct Node {
    char data; // 데이터가 들어가는 방을 생성합니다.
    struct Node* link; // 링크가 들어갈 방을 생성합니다.
} *top;

void Stack() {
    top = NULL;
}

int main() {

    Stack(); // 스택을 생성합니다.

    return 0;
}
```

```
Microsoft Visual Studio 디버그 콘솔

C:₩Users₩군인₩source₩repos₩Project9₩x64₩Debug₩Project9.exe(프로세스 33644개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

#define _CRT_NO_SECURE_WARNINGS

```
#include <stdio.h>
#include <malloc.h> //

struct Node {
	char data; // 데이터가 들어가는 방을 생성합니다.
	struct Node* link; // 링크가 들어갈 방을 생성합니다.
} * top;

void Stack() {
	top = NULL;
}

int main() {

	Stack(); // 스택을 생성합니다.

	return 0;
}
```

알게 된 내용: 연결 리스트를 이용하고자 하면 #include <malloc.h> 를 미리 선언해줘야 한다는 것과,
포인터변수를 활용할 줄 알아야 한다는 것을 배웠습니다.
또한, 배열을 이용할때는 StackLen 을 5로 설정하고 Arr[StackLen]으로 지정하여
처음부터 배열의 크기를 5로 지정해두고 시작했는데, 연결 리스트는 그렇지 않다, 즉 프로그램이 시작하면 저장공간을 만드는
동적 기억장소 이라는 것을 발견하였고,
이러한 배경 지식들을 확립하여 구조체를 생성함에 있어 무슨 원리로 만들어지는지를 알 수 있게 되었습니다.

2) IsEmpty() 를 생성, 이용하여 스택의 공백 여부를 확인해보자.

구현 및 구현 모습 :

```c
#define _CRT_NO_SECURE_WARNINGS
#include <stdio.h>
#include <malloc.h> //

struct Node {
    char data; // 데이터가 들어가는 방을 생성합니다.
    struct Node* link; // 링크가 들어갈 방을 생성합니다.
} *top;

void Stack() {
    top = NULL;
}

int IsEmpty() {
    if (top == NULL)
        return 1;
    else
        return 0;
}

int main() {
    Stack(); // 스택을 생성합니다.
    if (IsEmpty())
        printf("공백 상태 입니다.");
    else
        printf("공백 상태가 아닙니다.");

    return 0;
}
```

```
Microsoft Visual Studio 디버그 콘솔
공백 상태 입니다.
C:₩Users₩군인₩source₩repos₩Project9₩x64₩Debug₩Project9.exe(프로세스 29220개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

#define _CRT_NO_SECURE_WARNINGS
#include <stdio.h>

```
#include <malloc.h> //

struct Node {
	char data; // 데이터가 들어가는 방을 생성합니다.
	struct Node* link; // 링크가 들어갈 방을 생성합니다.
} * top;

void Stack() {
	top = NULL;
}

int IsEmpty() {
	if (top == NULL)
		return 1;
	else
		return 0;
}

int main() {
	Stack(); // 스택을 생성합니다.
	if (IsEmpty())
		printf("공백 상태 입니다.");
	else
		printf("공백 상태가 아닙니다.");

	return 0;
}
```

알게 된 내용 : IsEmpty() 의 구조는 배열을 이용한 스택에서의 구조와 동일하나, IsEmpty() 내의 if 문에서 top 이 NULL 이면
더 이상 저장공간을 생성 불가능, 으로 표현한다는 점이 배열과 다르다는 것을 알 수 있었습니다.

3) push(data e)를 이용하여 창고에 값을 집어넣어 보자.
구현 및 구현 모습 :

```
void push(char e) {
    struct Node* p = NULL; // 도우미 변수를 사용하여  기억 장소를 먼저 얻어냅니다.
    p = (struct Node*)malloc(sizeof(struct Node)); // p의 기억 장소를 생성합니다.
    if (p == NULL) { // 기억 장소를 생성했지만 NULL 이라면,
        return;
        printf("₩n삽입불가 ! 포화상태입니다."); // 더 이상 삽입할 수 없다, 라는 것을 알려줍니다.
    }
    else {
        p->data = e; // p가 가리키는 데이터에 e 값을 대입합니다.
        p->link = top; // p가 가리키는 링크에 top을 대입합니다.
        top = p; // top과 p는 같은 위치 입니다.
        showStackData();
    }
}

int main() {
    Stack(); // 스택을 생성합니다.
    int num;
    char val;

    while (1) {
        do {
            printf("₩n1.삽입 2.삭제 3.탐색 4.종료");
            scanf_s("%d", &num);
            while (getchar() != '₩n');
        } while (num < 1 || num>3);
        switch (num) {
        case 1:printf("₩n삽입할 값은?");
            scanf_s("%c", &val);
            push(val);
            break;
        }
    }
    return 0;
}
```

문제가 검색되지 않음

```
C:₩Users₩군인₩source₩repos₩Project9₩x64₩Debug₩Project9.exe

1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?a
<Stack> : a
1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?b
<Stack> : ba
1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?c
<Stack> : cba
1.삽입 2.삭제 3.탐색 4.종료
```

```
#define _CRT_NO_SECURE_WARNINGS
#include <stdio.h>
#include <malloc.h> //

struct Node {
	char data; // 데이터가 들어가는 방을 생성합니다.
	struct Node* link; // 링크가 들어갈 방을 생성합니다.
} * top;

void Stack() {
	top = NULL;
}

int IsEmpty() {
	if (top == NULL)
		return 1;
	else
		return 0;
}

void showStackData() {
	struct Node* p = NULL; // 도우미 변수를 사용하여 기억 장소를 먼저 얻어냅니다.
	p = top;
	printf("<Stack> : ");
	for (p; p != NULL;) { // p가 NULL이 아닌 동안,
		printf("%c", p->data); // p가 가리키는 데이터를 출력합니다.
		p = p->link; // p가 가리키는 link 값을 대입합니다.
	}
}

void push(char e) {
	struct Node* p = NULL; // 도우미 변수를 사용하여  기억 장소를 먼저 얻어냅니다.
	p = (struct Node*)malloc(sizeof(struct Node)); // p의 기억 장소를 생성합니다.
	if (p == NULL) { // 기억 장소를 생성했지만 NULL 이라면,
		return;
		printf("₩n삽입불가 ! 포화상태입니다."); // 더 이상 삽입할 수 없다, 라는 것을 알려줍니다.
	}
	else {
		p->data = e; // p가 가리키는 데이터에 e 값을 대입합니다.
		p->link = top; // p가 가리키는 링크에 top을 대입합니다.
		top = p; // top과 p는 같은 위치 입니다.
		showStackData();
	}
}

int main() {
	Stack(); // 스택을 생성합니다.
	int num;
	char val;

	while (1) {
		do {
			printf("₩n1.삽입 2.삭제 3.탐색 4.종료");
			scanf_s("%d", &num);
			while (getchar() != '₩n');
		} while (num < 1 || num>3);
			switch (num) {
			case 1:printf("₩n삽입할 값은?");
				scanf_s("%c", &val);
				push(val);
				break;
			}
	}
	return 0;
}
```

알게 된 내용 : push() 는 예상치 못하게, 배열을 이용한 스택과는 형태가 많이 다른 것을 느

꼈습니다.
역시 포인터 변수를 사용하다보니 연결리스트 에서는 제가 값을 넣을 장소를 직접 세세하게 지정하여
주어야 한다는 것이 큰 차이점인 것 같습니다.
여기서 부터 도우미 변수로 지정한 struct Node* p = NULL; 를 본격적으로 활용해 볼 수 있었습니다.
(struct Node*)malloc(sizeof(struct Node)); 에서는, malloc(sizeof(struct Node)를 활용하여
struct Node 타입의 기억장소의 크기를 알아내어 활용하는 방법을 알게 되었습니다.

4) pop() 을 생성, 이용하여 창고의 맨 위에 있는 물건을 꺼내보자.
구현 및 구현 모습 :

```c
int main() {
    Stack(); // 스택을 생성합니다.
    int num;
    char val;
    char ch;

    while (1) {
        do {
            printf("₩n1.삽입 2.삭제 3.탐색 4.종료");
            scanf_s("%d", &num);
            while (getchar() != '₩n');
        } while (num < 1 || num>4);
        switch (num) {
        case 1:printf("₩n삽입할 값은?");
            scanf_s("%c", &val);
            push(val);
            break;
        case 2:
            printf("₩n%c를 삭제합니다.", pop());
            break;
        case 4: return 0;
        }
    }
}
```

```
C:₩Users₩군인₩source₩repos₩Project9₩

1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?a
<Stack> : a
1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?b
<Stack> : ba
1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?c
<Stack> : cba
1.삽입 2.삭제 3.탐색 4.종료2
<Stack> : ba
c를 삭제합니다.
1.삽입 2.삭제 3.탐색 4.종료2
<Stack> : a
b를 삭제합니다.
1.삽입 2.삭제 3.탐색 4.종료2
<Stack> :
a를 삭제합니다.
1.삽입 2.삭제 3.탐색 4.종료
```

```
#define _CRT_NO_SECURE_WARNINGS
#include <stdio.h>
#include <malloc.h> //

struct Node {
	char data; // 데이터가 들어가는 방을 생성합니다.
	struct Node* link; // 링크가 들어갈 방을 생성합니다.
} *top;

void Stack() {
	top = NULL;
}

int IsEmpty() {
	if (top == NULL)
		return 1;
	else
		return 0;
}

void showStackData() {
	struct Node* p = NULL; // 도우미 변수를 사용하여 기억 장소를 먼저 얻어냅니다.
	p = top;
	printf("<Stack> : ");
	for (p; p != NULL;) { // p가 NULL이 아닌 동안,
		printf("%c", p->data); // p가 가리키는 데이터를 출력합니다.
		p = p->link; // p가 가리키는 link 값을 대입합니다.
	}
}

void push(char e) {
	struct Node* p = NULL; // 도우미 변수를 사용하여  기억 장소를 먼저 얻어냅니다.
	p = (struct Node*)malloc(sizeof(struct Node)); // p의 기억 장소를 생성합니다.
	if (p == NULL) { // 기억 장소를 생성했지만 NULL 이라면,
		printf("₩n삽입불가 ! 포화상태입니다."); // 더 이상 삽입할 수 없다, 라는 것을 알려줍니다.
	}
	else {
		p->data = e; // p가 가리키는 데이터에 e 값을 대입합니다.
		p->link = top; // p가 가리키는 링크에 top을 대입합니다.
		top = p; // top과 p는 같은 위치 입니다.
		showStackData();
	}
}

char pop() {
	struct Node* p = NULL; // 도우미변수 불러오기
	if (top == NULL) { // top 값이 NULL이면
		printf("삭제불가 ! 공백상태입니다."); // 겅벡 싱태임을 알려줍니다.
	}
	else {
		p = top; // top 값을 p에 대입
		top = top->link;
		char cha = p->data;
		free(p);
		showStackData();
		return cha;
	}
}

int main() {
	Stack(); // 스택을 생성합니다.
	int num;
	char val;
	char ch;

	while (1) {
		do {
			printf("₩n1.삽입 2.삭제 3.탐색 4.종료");
			scanf_s("%d", &num);
			while (getchar() != '₩n');
		} while (num < 1 || num>4);
		switch (num) {
		case 1:printf("₩n삽입할 값은?");
			scanf_s("%c", &val);
```

```
                        push(val);
                        break;
                case 2:
                        printf("₩n%c를 삭제합니다.", pop());
                        break;
                case 4: return 0;
                }
        }
}
```

알게 된 내용: pop() 은 먼저 push()를 생성해 보고 나니 더욱 쉽게 느껴졌습니다.
free() 라는 명령어를 알 수 있었고, 포인터 변수를 사용해 어디에 포인터 변수를 사용해 무엇이 무엇을 가리켜야 하는지 를 알 수 있었습니다.

5) peek()를 생성, 이용하여 창고의 맨 위에 있는 물건이 무엇인지 알아보자.

구현 및 구현 결과 :

```c
int main() {
    Stack(); // 스택을 생성합니다.
    int num;
    char val;
    char ch;

    while (1) {
        do {
            printf("₩n1.삽입 2.삭제 3.탐색 4.종료");
            scanf_s("%d", &num);
            while (getchar() != '₩n');
        } while (num < 1 || num>3);
        switch (num) {
        case 1:printf("₩n삽입할 값은?");
            scanf_s("%c", &val);
            push(val);
            break;
        case 2:
            printf("₩n%c를 삭제합니다.", pop());
            break;
        case 3:
            printf("₩n창고의 가장 위에 있는 값은 %c 입니다.", peek());
            break;
        case 4: return 0;
        }
    }
}
```

```
C:₩Users₩군인₩source₩repos₩Project9₩x64₩Debug

1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?a
<Stack> : a
1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?b
<Stack> : ba
1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?c
<Stack> : cba
1.삽입 2.삭제 3.탐색 4.종료1

삽입할 값은?d
<Stack> : dcba
1.삽입 2.삭제 3.탐색 4.종료3
<Stack> : dcba
창고의 가장 위에 있는 값은 d 입니다.
1.삽입 2.삭제 3.탐색 4.종료2
<Stack> : cba
d를 삭제합니다.
1.삽입 2.삭제 3.탐색 4.종료3
<Stack> : cba
창고의 가장 위에 있는 값은 c 입니다.
1.삽입 2.삭제 3.탐색 4.종료
```

#define _CRT_NO_SECURE_WARNINGS

```
#include <stdio.h>
#include <malloc.h>

struct Node {
	char data; // 데이터가 들어가는 방을 생성합니다.
	struct Node* link; // 링크가 들어갈 방을 생성합니다.
} * top;

void Stack() {
	top = NULL;
}

int IsEmpty() {
	if (top == NULL)
		return 1;
	else
		return 0;
}

void showStackData() {
	struct Node* p = NULL; // 도우미 변수를 사용하여 기억 장소를 먼저 얻어냅니다.
	p = top;
	printf("<Stack> : ");
	for (p; p != NULL;) { // p가 NULL이 아닌 동안,
		printf("%c", p->data); // p가 가리키는 데이터를 출력합니다.
		p = p->link; // p가 가리키는 link 값을 대입합니다.
	}
}

void push(char e) {
	struct Node* p = NULL; // 도우미 변수를 사용하여  기억 장소를 먼저 얻어냅니다.
	p = (struct Node*)malloc(sizeof(struct Node)); // p의 기억 장소를 생성합니다.
	if (p == NULL) { // 기억 장소를 생성했지만 NULL 이라면,
		printf("₩n삽입불가 ! 포화상태입니다."); // 더 이상 삽입할 수 없다, 라는 것을 알려줍니다.
	}
	else {
		p->data = e; // p가 가리키는 데이터에 e 값을 대입합니다.
		p->link = top; // p가 가리키는 링크에 top을 대입합니다.
		top = p; // top과 p는 같은 위치 입니다.
		showStackData();
	}
}

char pop() {
	struct Node* p = NULL; // 도우미변수 불러오기
	if (top == NULL) { // top 값이 NULL이면
		printf("삭제불가 ! 공백상태입니다."); // 공백 상태임을 알려줍니다.
	}
	else {
		p = top; // top 값을 p에 대입
		top = top->link;
		char cha = p->data;
		free(p);
		showStackData();
		return cha;
	}
}

char peek() {
	struct Node* p = NULL;
	if (top == NULL) { // 만약 top 값이 NULL 이면
		printf("공백상태 ! 창고에 값이 없습니다."); // 공백 상태임을 알려줍니다.
	}
	else {
		showStackData();
		return top->data; // top이 가리키는 데이터 값을 반환합니다.
	}
}
```

```
int main() {
	Stack(); //	스택을 생성합니다.
	int num;
	char val;
	char ch;

	while (1) {
		do {
			printf("₩n1.삽입 2.삭제 3.탐색 4.종료");
			scanf_s("%d", &num);
			while (getchar() != '₩n');
		} while (num < 1 || num>3);
		switch (num) {
		case 1:printf("₩n삽입할 값은?");
			scanf_s("%c", &val);
			push(val);
			break;
		case 2:
			printf("₩n%c를 삭제합니다.", pop());
			break;
		case 3:
			printf("₩n창고의 가장 위에 있는 값은 %c 입니다.", peek());
			break;
		}
	}
}
```

알게 된 내용: peek() 은 배열을 이용한 스택 생성에서도 다소 빠르게 이해했다 보니,
다행히도 연결 리스트를 이용한 스택에서도 문제없이 이해했습니다.
다만 포인터 변수에 대해서 더욱 공부해야 할 것 같습니다.

6) size()로 배열의 크기 알아내기, clear() 로 배열을 초기화 시키고 프로그램을 완성시키자.

구현 및 구현 결과 :

```
int main() {
    Stack(); // 스택을 생성합니다.
    int num;
    char val;
    char ch;

    while (1) {
        do {
            printf("₩n1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료");
            scanf_s("%d", &num);
            while (getchar() != '₩n');
        } while (num < 1 || num>6);
        switch (num) {
        case 1:printf("₩n삽입할 값은?");
            scanf_s("%c", &val);
            push(val);
            break;
        case 2:
            printf("₩n%c를 삭제합니다.", pop());
            break;
        case 3:
            printf("₩n창고의 가장 위에 있는 값은 %c 입니다.", peek());
            break;
        case 4:
            printf("크기는 %d 입니다.", size());
            break;
        case 5:
            printf("값을 초기화합니다.");
            clear();
            break;
        case 6:
            printf("프로그램을 종료합니다.");
            return 0;
        }
    }
};
```

문제가 검색되지 않음

검색 심도: 3

값

```
C:₩Users₩군인₩source₩repos₩Project9₩x64₩Debug₩Project9

1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료1

삽입할 값은?a
<Stack> : a
1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료1

삽입할 값은?b
<Stack> : ba
1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료4
크기는 2 입니다.
1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료5
값을 초기화합니다.<Stack> :
1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료1

삽입할 값은?a
<Stack> : a
1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료
```

```
#define _CRT_NO_SECURE_WARNINGS
#include <stdio.h>
#include <malloc.h>

struct Node {
	char data; // 데이터가 들어가는 방을 생성합니다.
	struct Node* link; // 링크가 들어갈 방을 생성합니다.
} * top;

void Stack() {
	top = NULL;
}

int IsEmpty() {
	if (top == NULL)
		return 1;
	else
		return 0;
}

void showStackData() {
	struct Node* p = NULL; // 도우미 변수를 사용하여 기억 장소를 먼저 얻어냅니다.
	p = top;
	printf("<Stack> : ");
	for (p; p != NULL;) { // p가 NULL이 아닌 동안,
		printf("%c", p->data); // p가 가리키는 데이터를 출력합니다.
		p = p->link; // p가 가리키는 link 값을 대입합니다.
	}
}

void push(char e) {
	struct Node* p = NULL; // 도우미 변수를 사용하여  기억 장소를 먼저 얻어냅니다.
	p = (struct Node*)malloc(sizeof(struct Node)); // p의 기억 장소를 생성합니다.
	if (p == NULL) { // 기억 장소를 생성했지만 NULL 이라면,
		printf("₩n삽입불가 ! 포화상태입니다."); // 더 이상 삽입할 수 없다, 라는 것을 알려줍니다.
	}
	else {
		p->data = e; // p가 가리키는 데이터에 e 값을 대입합니다.
		p->link = top; // p가 가리키는 링크에 top을 대입합니다.
		top = p; // top과 p는 같은 위치 입니다.
		showStackData();
	}
}

char pop() {
	struct Node* p = NULL; // 도우미변수 불러오기
	if (top == NULL) { // top 값이 NULL이면
		printf("삭제불가 ! 공백상태입니다."); // 공백 싱태임을 알려줍니다.
	}
	else {
		p = top; // top 값을 p에 대입
		top = top->link;
		char cha = p->data;
		free(p);
		showStackData();
		return cha;
	}
}

char peek() {
	struct Node* p = NULL;
	if (top == NULL) { // 만약 top 값이 NULL 이면
		printf("공백상태 ! 창고에 값이 없습니다."); // 공백 상태임을 알려줍니다.
	}
	else {
		showStackData();
		return top->data; // top이 가리키는 데이터 값을 반환합니다.
```

```
        }
}

int size() {
        struct Node* p = NULL; // 도우미 변수 불러오기
        int si = 0; // size 기본값을 설정합니다.
        p = top; // p에 top 값을 복사합니다.
        while (p != NULL) { // p가 NULL 값이 아니면 계속 반복합니다.
                si++; // size 값을 1씩 더합니다.
                p = p->link; //  p에 p가 가리키는 링크의 값을 복사합니다.
        }
        return si;
}

void clear() {
        struct Node* p = NULL; // 도우미 변수 불러오기.
        while (top != NULL) { // top 값이 NULL 값이 아니면 아래의 코드들을 계속 반복합니다.
                p = top; // top 값을 p 값에 복사합니다.
                top = top->link; // top이 가리키는 링크의 값을 top에 복사합니다.
                free(p); // p를 삭제합니다.
        }
        showStackData();
}


        int main() {
                Stack(); //      스택을 생성합니다.
                int num;
                char val;
                char ch;

                while (1) {
                        do {
                                printf("₩n1.삽입 2.삭제 3.탐색 4.크기 5.초기화 6.종료");
                                scanf_s("%d", &num);
                                while (getchar() != '₩n');
                        } while (num < 1 || num>6);
                        switch (num) {
                        case 1:printf("₩n삽입할 값은?");
                                scanf_s("%c", &val);
                                push(val);
                                break;
                        case 2:
                                printf("₩n%c를 삭제합니다.", pop());
                                break;
                        case 3:
                                printf("₩n창고의 가장 위에 있는 값은 %c 입니다.", peek());
                                break;
                        case 4:
                                printf("크기는 %d 입니다.",size());
                                break;
                        case 5:
                                printf("값을 초기화합니다.");
                                        clear();
                                        break;
                        case 6:
                                printf("프로그램을 종료합니다.");
                                return 0;
                        }
                }
        }
```

알게 된 내용: size() 와 clear() 은 배열을 이용한 스택에서는 사용하지 않았기 때문에 새로웠

습니다.
size() 에서는 while 문을 사용하여 배열의 크기만큼 si를 1씩 증가시킨 뒤 크기에 도달하면 값을 출력하는 형태 라는 것을 배웠고,
clear() 에서도 마찬가지로 while을 사용하여, 배열이 공백이 될 때까지
free() 를 이용하여 값을 삭제시키는 형태를 알 수 있었습니다.

과제에 관련하여 하고 싶은 말 :

기본적인 프로그램의 뼈대는 다 비슷한 것 같지만, 이렇게 연결 리스트로 만든
스택에서는 포인터 변수, malloc.h, sizeof 등
생소하다 느끼는 것들이 배열로 만든 스택에서 보다 많은 상태로
시작했기 때문에 숙지에 조금 시간이 걸릴 것 같다 라는 생각이 들었습니다.
특히 포인터 변수 같은 경우에는 제가 아직 포인터 변수 그 자체를
제대로 숙지하지 못하였기 때문에 본 과제에서 프로그램을 제작 하는 데에
더욱 어려움을 느꼈습니다. 앞으로 돌아가 포인터 변수를 제대로 다시
공부하고 다시 프로그램을 만들어보며 반복숙달 시킬 생각입니다.
과제 확인 해 주시느라 고생 많으셨습니다, 감사합니다 !
201932030 송현교 제출