Stack

동적할당을 이용한 스택 자료구조를 설계

학과	기계공학부
학번	200721395
이름	한경수
분반	001

- 다음과 같은 스펙을 만족시키는 프로그램을 짜라는 미션이 주어졌다.
- 스택이란?
  - ▫ 나중에 입력된 자료가 최상위에 위치하며 핚번에 최상위 데이터 하나를 읽어올 수 있다.
- 프로그램 기본 명령어
  - pushi – 스택에 정수 삽입(-2147483648 ~ 2147483647)
  - pushc – 스택에 문자열 삽입(20 자)
  - pushs – 스택에 긴 문자열 삽입(unlimited)
  - add – 스택의 내용 합치기
  - top – 최상위 스택 내용 보여주기
  - pop – 최상위 스택 내용 삭제
  - show – 스택의 모든 내용 보여주기
  - rev – 스택 배열 reverse
  - clear – 스택 비우기
  - quit – 프로그램 종료
- 위와 같은 상황을 만족시키는 프로그램을 짜기 위해서 linked list 구조를 다음과 같이 하였다.
  - typedef struct _LinkedStack
  - {
  - Data data;
  - struct _LinkedStack *prev;
  - struct _LinkedStack *next;
  - }
  - 다소 수업시간과는 생소한 구조일 수 있는데, 설명하자면 Data 는 어떤 자료를 담기 위한 임의의 structure 라고 생각하면 되고, 아래 두줄에 보이는 prev 와 next 는 다음 자료를 가르키는 포인터다. 보통 next 만 쓰는데, 여기서는 prev 까지 두었다.
  - rev 같은 명령어나 여러가지 다양한 접근을 하기 위해 이전 포인터까지 두었다.
  - LinkedStack* Head;
  - LinkedStack* Tail;
  - 또한 위와 같은 두개의 포인터를 둬서 바로 끝과 처음을 알 수 있게 설계하였다.
- 담을 수 있는 데이터는 정수형과 문자열이다. 다음과 같이 구조체를 두었다.
  - typedef struct
  - {
  - int digit;
  - char* longstr;
  - }Data
  - pushc 명령어와 pushs 명령어의 차이를 두기 위해 처음에는 char* longstr, char shortstr[20] 을 뒀지만, pushc 자체로도 add 에 의해 40 자가 넘어갈 수 있다는 답변을 듣고 그냥 범용적으로 char* 하나만 썼다. 하지만 이름에서 알 수 있다시피 longstr 라고 되어 있다. 범용적으로 긴 문자열도 담을 수 있다는 뜻이지, 꼭 긴 문자열을 받겠다는 의미는 아니다.
- 전체 프로그램구조는 main 함수에서 입력을 받고 그에 따른 subprogram 을 호출하는 식으로 만들었다.
  - 의사코드로 표현하면 다음과 같다.
  - main(){
    - while 1
      - str = input();
      - if(str == pushi) … pushi()
      - elif str == pushs … pushs()
      - elif str == add … add()
  - 그리고 예외처리를 확실히 하기 위해서 따로 스트림에서 입력받는 함수를 만들었다.
  - ReadLine 함수인데,
  - void ReadLine(char* dest, int maxcount)
  - {
  - char* temp = (char*)malloc(5000); // 일반적인 4096 보다 큼
  - memset(temp, 0, sizeof(char)*5000);
  - fgets(temp, 5000, stdin);
  -

- 	if(strlen(temp) != 0)
- 		temp[strlen(temp)-1] = '\0';
- 	strncpy(dest, temp, maxcount);
- 	dest[maxcount] = '\0';
- 	free(temp);
- }
- stream 의 버퍼는 크기는 4096byte 임을 생각해서 일단 5000byte 를 충분히 할당하고, fgets 함수를 통해 5000 개 까지 \n 가 있을 때 까지 모두 받아온다. 그리고 dest 에 strncpy 를 통해서 maxcount 만큼만 복사를 한다. 물론 세부적인 예외처리는 ReadLine 을 호출한 주체쪽에서 하게 될 것이다.
- 제 프로그램은 전체적으로 다 특별할거 없는데 rev 가 특별하다고 생각되어 rev 함수에 대해서 설명을 하고 이 보고서를 끝내겠습니다.
- List 순서를 뒤집는 rev 함수를 어떻게 구현할까 고심을 했습니다. 모두 어떠한 배열에 값을 저장한 후 다시 역으로 push 하는 방법은 어떨까 생각도 했다. 또, 내부적으로 시퀀스 규칙을 저장하는 변수를 만들어서 구현할까 생각도 했다. 전자의 문제는 overhead 가 너무 크다는 점이고, 후자인 경우에는 범용성이 매우 떨어진다는 단점이 있었다. 고심하던 찰나 prev pointer 를 쓰면 해결 될 것이라 생각이 이르렀다. 기본 아이디어는 각 노드의 next 와 prev 를 전부 swap 한다. 그리고 tail 과 head 를 swap. 그렇게 한다면 list 가 뒤집어진다. 명확했다. 프로그램 소스는 이 문서에 첨부첨부하는 것 보단 따로 첨부하는게 나을거 같아서 따로 올리겠다.