[시스템프로그래밍기초 팀 프로젝트 일지\_4회차]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 일시 | 2019년 11월 21일 13:00 ~ 15:00 | | | |
| 팀원 | 안세윤 | 2019064548 | 윤성빈 | 2019082206 |
| 이다은 | 2019036380 | 임단비 | 2019056462 |
| 토의내용 및 소스코드 구현 | **과제 요건 <스택을 이용한 수식 표기법 변환 함수를 구현해야 한다.> 을 만족하기 위해 필요한 개념 및 알고리즘 & 코드 구현**  **<스택(stack>)**   1. 스택의 정의 : 제한적으로 접근할 수 있는 나열 구조로, 그 접근 방법은 언제나 목록의 끝에서만 일어난다. 스택은 한 쪽 끝에서만 자료를 넣거나 뺄 수 있는 선형 구조(LIFO, Last In First Out)로 되어 있다. 자료를 넣는 것을 ‘밀어 넣는다’하여 푸시(push)라 하고 반대로 넣어둔 자료를 꺼내는 것을 팝(pop)이라고 하는데, 이 때, 꺼내지는 자료는 가장 최근에 보관한 자료부터 나오게 된다. 이처럼 나중에 넣은 값이 먼저 나오는 것을 LIFO 구조라고 한다.     출처: http://ehpub.co.kr/3-3-%EC%8A%A4%ED%83%9Dstack-stl-stack-%EC%82%AC%EC%9A%A9/  위 그림에서,   * 초기 스택에 아무것도 넣지 않았을 때 empty인 상태를 top = -1로 표현한다. * Push(n)일 때, n을 스택에 가장 밑에 넣는다. * 이후 push(n)은 괄호 안의 수 n을 스택에 차곡차곡 쌓는다. * Pop()일 때, ‘후입 선출’의 개념을 이용하여 가장 마지막에 넣은, 즉 top의 수를 스택에서 꺼내고 꺼낸 수를 출력한다.  1. 코드 분석(stack.c 파일) 2. **operStk (연산자 관련) : 중위표기식을 후위표기식으로 바꾸는 과정에서 연산자를 넣고 빼며 후위표기식의 순서대로 만드는데 사용한다.**      * **operStkNode\* newOperStkNode(char c)**   : operStkNode\*를 타입으로 하고, char형 변수 c를 받아 새로운 operStkNode를 만든다. 이 노드는 연산자(operand)와 이전 operStkNode를 가리키는 포인터 prev를 갖는데, 처음에 prev는 NULL값으로 초기화 한다.   * **operStkNode\* tmp = (operStkNode\*)malloc(sizeof(operStkNode))**   : 새로 만들어진 노드를 가리키는 포인터는 tmp이고, 노드를 파일 cal.h에서. 구조체로 선언했던 operStkNode의 크기만큼malloc(메모리 동적 할당) 해 준다.   * **tmp->oper = c**   : tmp가 가리키는 노드의 data부분, oper에 함수의 인자로 받아온 char형 변수 c를 assign한다.   * **tmp->prev = NULL**   : tmp가 가리키는 노드의 prev 부분을 NULL로 초기화한다.   * **return tmp**   : 노드를 가리키는 포인터 tmp를 반환한다.     * **operStk\* newOperStk()**   : operStk\*를 타입으로 하는 연산자를 담는 스택을 선언하며, operStk은 중위. 표기식을 후위 표기식으로 바꿀 때 사용한다.   * **operStk\* tmp = (operStk\*)malloc(sizeof(operStk))**   : 스택을 가리키는 포인터 이름이 tmp이고, 이 스택을 파일 cal.h에서 구조체로. 선언했던 operStk의 크기만큼 malloc(메모리 동적 할당) 해준다.   * **tmp->top = NULL**   : tmp가 가리키는 스택의 top(가장 윗 부분)을 NULL로 초기화한다.   * **return tmp**   : 스택을 가리키는 포인터 tmp를 반환한다.     * **void push2Ostk(operStk\* stk, char c)**   : operStk\*의 타입을 가지는 스택 stk에서char형 변수 c를 받아void 형을. 리턴하고, 스택에 연산자를 push하는 함수를 선언한다.   * **operStkNode\* tmp = newOperStkNode(c)**   : 변수 c를 받고operStkNode\*형을 가지며 노드를 가리키는 새로운 포인터 tmp를. 선언한다.   * **tmp->prev = stk->top**   : tmp가 가리키는 노드의 prev가 스택 stk의 top을 가리키도록 한다. 즉, prev가. stk의 top의 주소값을 가진다. 스택에서는 이중 연결리스트가 아닌 단일연결리스트의 개념을 사용한다.   * **stk->top = tmp**   : 스택 stk의 top은 tmp가 가리키는 노드가 된다.     * **char pop4Ostk(operStk\* stk)**   : operStk\* 형을 가지는 스택 stk를 인자로 받아 스택의 가장 위 연산자를 pop하여. char형인 그 문자를 반환하는 함수를 선언한다.   * **operStkNode\* tmp = stk->top**   : operStkNode\*형을 가지는 노드를 가리키는 tmp가 스택 stk의 top의 주소값을. 가진다.   * **char poped = tmp->oper**   : 자료형이 char형인 변수 poped가 tmp가 가리키는 노드의 oper부분을. 가리키도록 한다.   * **stk->top = tmp->prev**   : 스택 stk의 top은 tmp가 가리키는 노드의 prev, 즉, 스택 상의 tmp가 가리켰던. 노드 아래에 있던 노드가 된다.   * **free(tmp)**   : 메모리 동적 할당 해주었던 노드를 free해줌으로써 동적 할당 해제를 해준다.   * **return poped**   : 변수 poped에 저장된 (top이었던) 노드의 oper를 반환한다.     * **int emptyOstk(operStk\* stk)**   : operStk\*의 타입을 가지는 스택 stk을 인자로 받아 스택이 비어있는지 아닌지. int형의 결과값으로 보여주는 함수   * **if(stk->top == NULL) return 1**   : 만약 스택 stk의 top이 NULL일 경우, 즉 스택이 비어있을 경우 숫자 1을. 반환한다.   * **else return 0**   : 만약 스택 stk의 top이 NULL이 아닐 경우, 즉 스택이 비어있지 않을 경우 숫자. 0을 반환한다.   1. **numStk (피연산자 관련) : 최종 계산 과정에서 NUM 형 포인터를 넣고 빼면서 계산할 때 사용 한다.**      * **numStkNode\* newNumStkNode(NUM \*n)**   : numStkNode\*를 타입으로 하고, NUM\*형 변수 n(피연산자)을 인자로 받는 새로운. 노드 함수를 선언한다.   * **numStkNode\* tmp = (numStkNode\*)malloc(sizeof(numStkNode))**   : numStkNode\*형이고 tmp가 가리키는 노드를 선언하고, 이 노드를. numStkNode의 크기만큼 malloc(메모리 동적 할당) 해준다.   * **tmp->num = n**   : tmp가 가리키는 노드의 num에 받은 인자 n을 assign한다.   * **tmp->prev = NULL**   : tmp가 가리키는 노드의 prev를 NULL로 초기화한다.   * **return tmp**   : 노드를 가리키는 포인터 tmp를 반환한다.     * **numStk\* newNumStk()**   : numStk\*형을 타입으로 가지는 새로운 스택을 선언한다. 이 스택은 numNode를. 담으며, 최종 계산 과정에서 NUM형 포인터를 넣고 빼면서 계산할 때 사용한다.   * **numStk\* tmp = (numStk\*) malloc(sizeof(numStkNode))**   : numStk\*타입의 스택을 가리키는 포인터 tmp를 선언하고, 이를 numStkNode의. 크기만큼 malloc(메모리 동적 할당) 해준다.   * **tmp->top = NULL**   : tmp가 가리키는 스택의 top을 NULL로 초기화한다.   * **return tmp**   : 스택을 가리키는 포인터 tmp를 반환한다.     * **void push2Nstk(numStk \*stk, NUM \*n)**   : numStk\*의 타입을 가지는 스택 stk에서 NUM\*형 변수 n을 받아 void형을. 리턴하고, 스택에 연산자를 push하는 함수를 선언한다.   * **numStkNode\* tmp = newNumStkNode(n)**   : 변수 n을 받고 numStkNode\*형을 가지며 포인터 tmp가 가리키는 새로운 노드를. 선언한다.   * **tmp->prev = stk->top**   : tmp가 가리키는 노드의 prev가 스택 stk의 top을 가리키도록 한다. 즉, 노드의. prev가 stk의 top의 주소값을 가진다. 스택에서는 이중 연결리스트가 아닌 단일연결리스트의 개념을 사용한다.   * **stk->top = tmp**   : 스택 stk의 top은 tmp가 가리키는 노드가 된다.     * **NUM\* pop4Nstk(numStk \*stk)**   : numStk\* 형을 가지는 스택 stk을 인자로 받아 스택의 가장 위 피연산자를. pop하여 NUM\*형인 그 수를 반환하는 함수를 선언한다.   * **numStkNode\* tmp = stk->top**   : numStkNode\* 형을 가지고 tmp가 가리키는 노드는 스택 stk의 top의 주소값을. 가진다.   * **NUM\* poped = tmp->num**   : 자료형이 NUM\* 형인 변수 poped가 노드의 num 부분을 가리키도록 한다.   * **stk->top = tmp->prev**   : 스택 stk의 top은 노드의 prev, 즉, 스택 상의 노드 아래에 있던 노드가 된다.   * **free(tmp)**   : 메모리 동적 할당을 해주었던 노드를 free해줌으로써 동적 할당 해제 해준다.   * **return poped**   : 변수 poped에 저장된 (top이었던) 노드의 num을 반환한다.     * **int emptyNstk(numStk \*stk)**   : numStk\*의 타입을 가지는 스택 stk을 인자로 받아 스택이 비어있는지 아닌지 int. 형의 결과값으로 보여주는 함수   * **if(stk->top == NULL) return 1**   : 만약 스택 stk의 top이 NULL일 경우, 즉 스택이 비어있을 경우 숫자 1을. 반환한다.   * **else return 0**   : 만약 스택 stk의 top이 NULL이 아닐 경우, 즉 스택이 비어있지 않을 경우 숫자. 0을 반환한다. | | | |
| 결론 | 스택은 마치 하단이 막혀 있고 상단이 뚫려 있는 좁은 상자와도 같다. 즉, 우리가 흔히 볼 수 있는 ‘선입선출’과 달리 ‘후입선출’을 이용하는데, 접해보지 않은 사람들에게는 생소한 개념이다. 스택은 큐나 연결리스트에 비해 구현하기가 비교적 쉽지만, 스택을 처음 접하는 사람들이 이해하기에는 시간이 조금 걸리며 진입장벽이 낮지만은 않은 개념인 것 같다. 실제로 이 코드를 구현할 때, 스택에 관한 내용을 이해하는데 시간이 조금 소요되었으며, 스택을 노드 및 연결리스트와 연관지어 생각하기 쉽지 않았다. 또한, 스택이 있어 연산자와 피연산자를 보관하고 계산하기에 용이한 것은 틀림없다는 것을 알게 되었고, 스택을 노드와 연관지어 구현해봄으로써 메모리 동적 할당과 연결리스트, 그리고 구조체 포인터 안의 구조체 변수에 쉽게 접근하는 “->” 에 대해 깊이 연구해보는 계기가 되었다. | | | |