자료구조 과제 5

* –

20151523

김동현

- 실행결과



input

시계, 그리기이(가) 표시된 사진

자동 생성된 설명

Output

- 알고리즘 및 코드 설명

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_EXPR\_SIZE 100

typedef enum {lparen, rparen, plus, minus, times, divide, mod, eos, operand}

precedence;

//precedence \*stack;

//int top = -1;

char expr[MAX\_EXPR\_SIZE];

int isp[] = {0, 19, 12, 12, 13, 13, 13, 0}; // 교과서 3.6절 pp. 136 참고

int icp[] = {20, 19, 12, 12, 13, 13, 13, 0};

: 각 연산기호 및 피연산자에 대한 enumerated value의 정의를 해주고 precedence라는 type으로 명명한다. 그리고 node라는 type을 새로 정의하여서 그 type의 linked-list 방식으로 stack을 구성할 것이기 때문에 주어진 precedence \*stack 과 int top은 사용하지 않았다. 그리고 expr에는 expr.txt를 읽은 문자를 저장해주는 배열이고, 연산자들 간의 우선도를 결정지을 isp(in-stack precedence)와 icp(incoming precedence)를 세팅하였다.

typedef struct node\* nodepointer;

typedef struct node {

precedence token;

nodepointer link;

}node;

nodepointer top;

: linked list의 각 노드에는 token이라는 멤버가 들어있고, 이 멤버에 연산기호의 enum이 저장될 것이다. 그리고 link를 통해 다음 노드로 나아갈 수 있게 정의하였다. Top를 선언하여 스택의 현재 최상단 노드를 표시하게 하였다.

void push(precedence item) {

nodepointer temp = (nodepointer)malloc(sizeof(node));

temp->token = item;

temp->link = top;

top = temp;

}

: stack의 push 함수이다. temp라는 새로운 노드를 함수내에서 선언하여 그 temp 노드에 최근 연산자를 저장하고, link에 top 노드를 가리키게끔 하였다. 그리고 그 temp가 이제 stack의 최상단 노드가 되었으므로 top를 temp로 초기화해주었다. 위와 같은 방식으로 설정하였으므로 linked-list의 구조는 stack이면서 각 노드들은 바로 아래의 노드를 가리키게 되는 순서로 설계한 것이다.

precedence pop() {

nodepointer temp = (nodepointer)malloc(sizeof(node));

precedence element;

temp = top;

element = temp->token;

top = temp->link;

free(temp);

return element;

}

: push와 반대로 pop은 우선 임시변수 element에 top의 token을 저장하고, 그 노드를 해제하면서 그 노드가 가리키던 노드를 top으로 초기화 해준다. 그리고 그 element에 저장했던 enum을 반환한다.

void init(nodepointer a) {

a = (nodepointer)malloc(sizeof(node));

a->link = NULL;

}

: linked-list를 처음 구성할 때, 첫 노드의 link값을 NULL로 초기화 시켜주었다.

precedence getToken(char\* symbol, int\* n) {

/\* get the next token, symbol is the character representation, which is returned, the

token is represented by its enumerated value, which is returned in the function name \*/

\*symbol = expr[(\*n)++];

switch (\*symbol) {

case '(': return lparen;

case ')': return rparen;

case '+': return plus;

case '-': return minus;

case '/': return divide;

case '\*': return times;

case '%': return mod;

case '\0': return eos;

default: return operand; /\* no error checking, default is operand \*/

}

}

: 파일에서 읽어온 문자가 들어있는 expr 배열에서 문자 한 개를 받아와서 enum으로 변환해주는 함수이다. 아래 postfix 함수에서 이 함수를 호출 시 symbol의 주소값을 매개변수로 전달하므로, 자동으로 symbol 변수에 expr의 원소가 저장되는 부분도 있다.

void printToken(precedence p) {

char a;

switch (p) {

case lparen: a = '(';

break;

case rparen: a = ')';

break;

case plus: a = '+';

break;

case minus: a = '-';

break;

case times: a = '\*';

break;

case divide: a = '/';

break;

case mod: a = '%';

break;

default: a = NULL;

}

printf(" %c", a);

}

: 매개변수로 받아온 연산기호 enum을 실제 문자로 출력해주는 함수이다.

void postfix() {

char symbol;

int n = 0;

int count = 0;

precedence token;

nodepointer temp = (nodepointer)malloc(sizeof(node));

temp->token = eos;

temp->link = top;

top = temp;

for (token = getToken(&symbol, &n); token != eos; token = getToken(&symbol, &n)) {

if (token == operand)

printf("%c", symbol);

else if (token == rparen) {

/\* unstack tokens until left paranthesis \*/

while (top->token != lparen)

printToken(pop());

pop(); /\* discard the left paranthesis \*/

}

else {

/\* remove and print symbols whose isp is greater

than or equal to the current token’s icp \*/

while (isp[top->token] >= icp[token])

printToken(pop());

push(token);

}

}

while ((token = pop()) != eos)

printToken(token);

printf("\n");

}

: getToken을 통해 token에 expr 배열에서 eos(\0)가 나올 때까지 문자를 분류한다.

만일 token이 피연산자인 경우, 그냥 출력한다. 만일 token이 ‘)’ 인 경우 ‘(‘ 까지의 원소를 stack에서 꺼내어 출력해준다. (a - b / c)인 경우라고 예를 들면 a,b,c는 이미 출력되어 있고, () 사이에 들어있는 연산기호 /, - 를 stack에서 pop하여 출력해준다. 그리고 마지막에 ‘(‘은 pop하여 버려준다.

만일 token이 연산 기호인 경우 stack의 최상단 token과 우선도를 따진다. 만일 stack의 top token보다 token의 우선도가 더 낮으면 stack에서 token보다 우선도가 낮은 연산기호가 나타날 때까지 pop하여 출력해준다. Eos를 만나 반복문을 탈출하면 마지막으로 stack에 남아있는 기호들을 pop하여 출력해준다.

전체적으로 봤을 때, 예를 들어 A+B\*C-D 라는 식이 있다면 (A 출력) -> (+ 와 초기의 eos 비교, +가 더 크므로 출력없이 push) -> (B 출력) -> (\*와 top의 token +와 비교, \*가 더 크므로 출력없이 push) -> (C 출력) -> (-와 top의 token \*와 비교, -가 더 작으므로 stack에서 pop하여 \*, + 순으로 출력 후 - push) -> (D출력) -> (eos를 만나 stack의 남은 기호 – 출력) 의 과정으로 프로그램이 실행된다.

void rExpr(FILE\* fp, char\* expr) {

int count = 0;

while (!(feof(fp)))

fscanf(fp, "%c", &expr[count++]);

}

: 단순히 fp 파일 포인터에 저장된 파일의 끝이 나올 때까지 한 글자씩 읽어들여서 expr배열에 차례대로 저장해주는 함수이다.

void main()

{

FILE\* fp = fopen("expr.txt", "r");

rExpr(fp, expr);

init(top);

postfix();

return 0;

}

: 파일 포인터에 해당 파일 지정 후 rExpr 함수를 호출하여 expr 배열에 문자들을 저장하고, init 함수를 통해 linked-list의 head node->link를 NULL로 초기화, 그 다음 postfix 함수를 호출하여 일련의 작업을 실행한 뒤 종료한다.