◇ 문제 및 요구하는 것

**1. 사용자로부터 수식을 입력받고, 이를 후위 식으로 변환하고, 후위 식을 평가하여 그 결과 값을 출력하는 프로그램을 작성하고 테스트하라**

**추가 사항 고려**

**- 수치를 2자리 이상일 경우**

**- 거듭제곱 연산자 ‘^’ 추가 (2^3 = 8)**

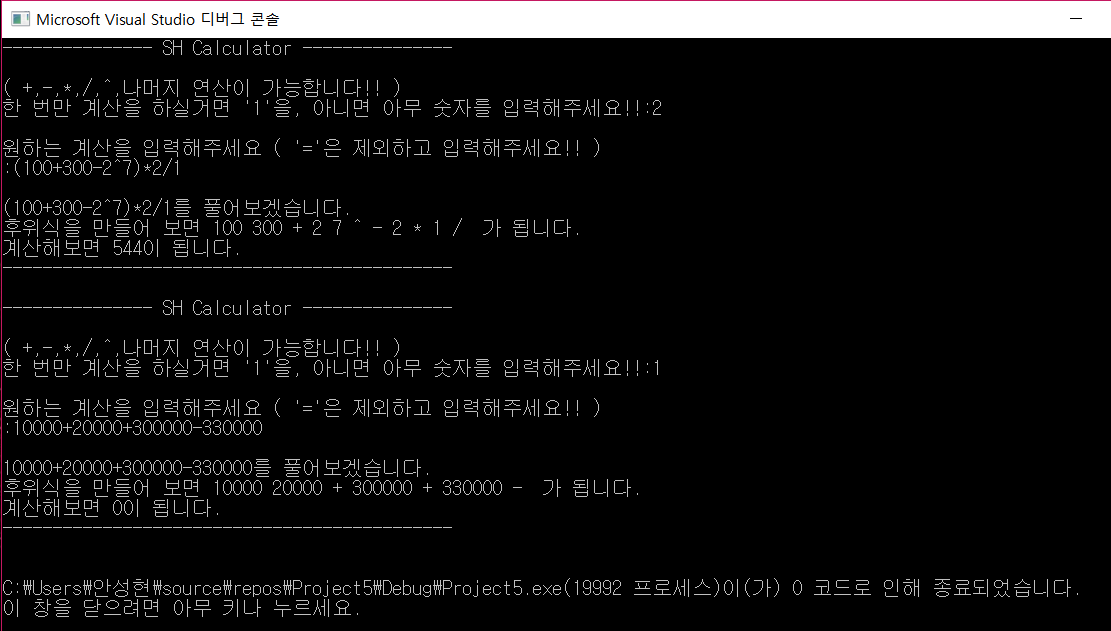
◇ 제목

◇ 제목

◇ 제목

◆ 프로그램 실행 결과

( 추가 사항 모두 적용 )



◆ IOPE 차트 만들기 (후위식 변환 기준)

|  |  |
| --- | --- |
| 차트 |  |
| **INPUT** | exp[max]: char (중위식 배열) , pexp[max]: int (후위식 배열) |
| **OUTPUT** | X (다만, 포인터를 이용하므로 배열 값이 저절로 바뀜) |
| **PROCESSING** | **while ((tmp <- exp배열의 문자를 하나씩 가져옴) != '\0') do**  **switch (tmp)**  **case '+': case '-': case '\*': case '/': case '%': case '(': case'^': case ')':**  if (tmp = ')') then **// ↓ tmp가 ‘)’ 일 때 처리**  while ((left = pop(&st)) != ‘(’) do  pexp[i++] <- left; // ‘(‘ 빠질 때까지 빼고 배열에 넣기  repeat  break; // switch문 빠져 나오기  endif **// ↓ tmp가 ‘)’ 아닐 때 처리**  while (!is\_empty(st) and (p <- 스택 최상위 원소) >= tmp ) do  if (p = ‘(‘ ) then // 스택 최상위 있는게 '('이고 우선순위가 더 높다면,  break; // '('은 ')'가 와야만 빠질 수 있으므로 break로 중단. '('밑에 쌓인 것은 볼 필요도 없음  else  pexp[i++] <- pop(&st);  endif  repeat  if (tmp = '+') then push(&st, -43); else if (tmp = '-') then push(&st, -45);  else if (tmp = '\*') then push(&st, -42); else if (tmp = '/') then push(&st, -47);  else if (tmp = '%') then push(&st, -37); else if (tmp = '^') then push(&st, -94);  else // tmp가 괄호일 때 push(&st, -40); endif break; // switch문 빠져 나오기  **// (After) 만약 tmp가 숫자이면 그냥 pexp(후위식) 배열에 넣기** |
| **EXAMPLE** | **If(exp[]=”2+3-5”)↓**  **1.** 처음 tmp=’2’ -> pexp[0]=2  **2.** ‘+’가 왔으니 while문으로 가서 스택이 비었는지 검사 -> 아직 스택 사용을 안 했으니 비어 있음 -> 스택에 ‘+’를 넣음  **3.** tmp=’3’이니 pexp[1]=3 **4.** ‘-‘가 왔는데 스택이 안 비었음 -> 스택 최상단의 ‘+’랑 우선 순위가 같으니 ‘+’를 스택에서 빼고 pexp배열에 넣음 -> ‘-‘를 스택에 넣음 **5.** pexp[3]=5 **6.** Processing에는 안 써있지만, 스택에 남은 연산자들은 모두 pexp배열에 넣음 |

◆ SPARKS 언어로 알고리즘 작성하기

**procedure postfix (exp[]: char, pexp[]: int)**

**for (i <- 0 to max-1) do**

// pexp함수를 쓰레기값으로 초기화 (초기화 안 하면 두 번째 실행 시, 첫 번째 실행할 때 저장한 요소가 영향을 끼침)

pexp[i] <- -100;

**repeat**

char: tmp; int: left, p; // 임시 변수

int: jnum <- 0; // 수의 자릿수 변수

int: n <- 0; // 현재 수를 저장한 변수

int: i <- 0; // pexp배열의 인덱스

**while ((tmp <- get\_symbol(exp, 0)) != '\0') do**

**switch (tmp)**

**case '+': case '-': case '\*': case '/': case '%': case '(': case'^': case ')':**

**if (jnum >= 1) then // 더이상 연속적으로 숫자가 나오지 않았을 때**

pexp[i++] <- n; // 수를 일단 배열에 넣음 (수를 먼저 처리하고 연산자를 처리해야 하므로)

jnum <- 0; n <- 0; // 이번 수 처리는 다 했으니, 다음 수 처리를 위해 jnum,n을 0으로

**endif**

**if (tmp = ')') then**

**while ((left = pop(&st)) != -40) do // '('나올 때까지 꺼내고 '('가 아니면 배열에 저장**

pexp[i++] <- left;

**repeat**

**break; // switch문 빠져 나오기**

**endif**

**while (!is\_empty(st) and (p <- PIS(peek(st))) >= PIE(tmp)) do**

**if (p = 4) then // 스택 최상위 있는게 '('이면**

break; // '('은 ')'가 와야만 빠질 수 있으므로 break로 중단. '('밑에 쌓인 것은 볼 필요도 없음

**else**

pexp[i++] <- pop(&st);

**endif**

**repeat**

**if (tmp = '+') then**

push(&st, -43);

**else if (tmp = '-') then**

push(&st, -45);

**else if (tmp = '\*') then**

push(&st, -42);

**else if (tmp = '/') then**

push(&st, -47);

**else if (tmp = '%') then**

push(&st, -37);

**else if (tmp = '^') then**

push(&st, -94);

**else // tmp가 괄호일 때**

push(&st, -40);

**endif**

**break; // switch문 빠져 나오기**

**default: // 그냥 숫자를 받았을 때**

n <- 10 \* n;

n <\_ n + (tmp - 48);

jnum++;

**if (jnum >= 1 and get\_symbol(exp, 1) = '\0') then // 연속으로 숫자를**

**받았는데, 다음 문자가 널 문자일 때!**

pexp[i++] <\_ n;

**endif**

**break; // switch문 빠져 나오기**

**while (!is\_empty(st)) do // 식이 끝나면 스택이 비어있을 때까지 빼네서 배열에 저장**

**if ((left = pop(&st)) != -40) then // ')'면 배열에 저장 안 함**

pexp[i++] <- left;

**endif**

**repeat**

**end procedure**

#include <stdio.h>

#define max 30 // 피연산자와 연산자의 개수

typedef struct {

int stack[max];

int top;

}stack\_type;

typedef stack\_type\* stack\_type\_ptr;

stack\_type st;

int stop,num,num2; // stop이 1되면 멈춤, num,num2는 get\_symbol에서 유용하게 사용됨

void init(stack\_type\_ptr s) {

s->top = -1;

}

int is\_full(stack\_type s) {

if (s.top == max - 1)

return 1;

else

return 0;

}

int is\_empty(stack\_type s) {

if (s.top == -1)

return 1;

else

return 0;

}

void push(stack\_type\_ptr s, int item) {

if (!is\_full(\*s)) {

s->top++;

s->stack[s->top] = item;

}

else

printf("스택이 꽉 차있음");

}

int pop(stack\_type\_ptr s) {

if (!is\_empty(\*s)) {

int tmp = s->stack[s->top];

s->top--;

return tmp;

}

else

printf("스택이 비어 있음");

}

int peek(stack\_type s) { // top을 줄이지 않고 스택 최상단에 있는 것을 출력

int tmp = s.stack[s.top];

return tmp;

}

int PIS(int alter) { // 스택 최상단에 위치한 연산자를 우선 순위에 맞추어 리턴

switch (alter) {

case -40: // 괄호가 들어왔을 때

return 4;

case -94: // ^가 들어왔을 때

return 3;

case -37: case -42: case -47: // %,\*,/ 가 들어왔을 때

return 2;

case -43: case -45: // +,-가 들어왔을 때

return 1;

}

}

int PIE(char alter) { // 식에 있는 연산자를 우선 순위에 맞추어 리턴

switch (alter) {

case '(': // 괄호가 들어왔을 때

return 4;

case '^': // ^가 들어왔을 때

return 3;

case '%': case '\*': case '/': // %,\*,/ 가 들어왔을 때

return 2;

case '+': case '-': // +,-가 들어왔을 때

return 1;

}

}

char get\_symbol(char \*exp, int sp) { //문자열 받아서 문자 한 개씩 보내기

if (sp == 0)

return exp[num++];

else if (sp == 1) // num에 1을 더하여 저장하면 안 되는 특별한 경우

return exp[num];

}

void postfix(char \*exp, int \*pexp) { // 후위식 만들기

for (int i = 0; i < max; i++) {

// pexp함수를 쓰레기값으로 초기화 (초기화 안 하면 두 번째 실행 시, 첫 번째 실행할 때 저장한 요소가 영향을 끼침)

pexp[i] = -100;

}

char tmp; int left, p; // 임시 변수

int jnum = 0; // 수의 자릿수 변수

int n = 0; // 현재 수를 저장한 변수

int i = 0; // pexp배열의 인덱스

while ((tmp = get\_symbol(exp, 0)) != '\0') {

switch (tmp) {

case '+': case '-': case '\*': case '/': case '%': case '(': case'^': case ')':

if (jnum >= 1) { // 더이상 연속적으로 숫자가 나오지 않았을 때

pexp[i++] = n; // 수를 일단 배열에 넣음 (수를 먼저 처리하고 연산자를 처리해야 하므로)

jnum = 0; n = 0; // 이번 수 처리는 다 했으니, 다음 수 처리를 위해 jnum,n을 0으로

}

if (tmp == ')') {

while ((left = pop(&st)) != -40) { // '('나올 때까지 꺼내고 '('가 아니면 배열에 저장

pexp[i++] = left;

}

break;

}

while (!is\_empty(st) && (p = PIS(peek(st))) >= PIE(tmp)) {

if (p == 4) // 스택 최상위 있는게 '('이면

break; // '('은 ')'가 와야만 빠질 수 있으므로 break로 중단. '('밑에 쌓인 것은 볼 필요도 없음

else

pexp[i++] = pop(&st);

}

if (tmp == '+')

push(&st, -43);

else if (tmp == '-')

push(&st, -45);

else if (tmp == '\*')

push(&st, -42);

else if (tmp == '/')

push(&st, -47);

else if (tmp == '%')

push(&st, -37);

else if (tmp == '^')

push(&st, -94);

else // tmp가 괄호일 때

push(&st, -40);

break;

default: // 그냥 숫자를 받았을 때

n = 10 \* n;

n = n + (tmp - 48);

jnum++;

if (jnum >= 1 && get\_symbol(exp, 1) == '\0') // 연속으로 숫자를 받았는데, 다음 문자가 널 문자일 때!

pexp[i++] = n;

break;

}

}

while (!is\_empty(st)) // 식이 끝나면 스택이 비어있을 때까지 빼네서 배열에 저장

if ((left = pop(&st)) != -40) // ')'면 배열에 저장 안 함

pexp[i++] = left;

}

int get\_symbol2(int \*pexp) { //후위식 받아서 (연산자or피연산자) 한 개씩 보내기

return pexp[num2++];

}

int power(int a, int b) { // 제곱 함수

if (b == 0)

return 1;

else

return a \* power(a, b - 1);

}

int exec(int tmp, int left, int right) { // tmp데이터에 따라 left와 right로 계산하는 함수

switch (tmp) {

case -43: // 더하기

return right + left;

case -45: // 빼기

return right - left;

case -42: // 곱하기

return right \* left;

case -47: // 나누기

return right / left;

case -37: // 나머지

return right % left;

case -94: // 제곱

return power(right, left);

}

}

int eval(int \*pexp) { // 최종 계산 값 리턴

int tmp, r; //임시 변수

int left, right; // 스택에서 꺼낸 값을 저장하는 변수

for (int i = 0; pexp[i] > -95; i++) { // -95부터 쓰레기 값이라고 간주함 -> 쓰레기 값이 아니면 계속 연산에 활용함

tmp = get\_symbol2(pexp);

switch (tmp) {

case -43: case -45: case -42: case -47: case -37: case -94: // tmp가 연산자일 때

left = pop(&st);

right = pop(&st);

r = exec(tmp, left, right);

push(&st, r);

break;

default: // tmp가 피연산자일 때

if (tmp >= 0)

push(&st, tmp);

break;

}

}

return st.stack[st.top]; // 연산이 잘 됐으면 top(0)일 때의 스택 값이 최종 값임

}

void display(char\* exp, int\* pexp, int res) {

// 원래식 출력

printf("\n%s를 풀어보겠습니다.", exp);

// 후위식 출력

printf("\n후위식을 만들어 보면 ");

for (int i = 0; pexp[i] > -95; i++) { // -95부터 쓰레기 값이라고 간주함

if (pexp[i] >= 0) // 피연산자일 때

printf("%d ", pexp[i]);

else // 연산자일 때

printf("%c ", -(pexp[i])); // 연산자가 '-(아스키코드)' 형태로 저장되었으니, 다시 -를 해서 문자로 출력

}

printf(" 가 됩니다.");

// 결과 출력

printf("\n계산해보면 %d이 됩니다.\n", res);

printf("---------------------------------------------\n\n");

}

void get\_exp(char \* exp) { // 사용자로부터 중위식(원래식)을 입력받음

for (int i = 0; i < max; i++) {

// exp,pexp함수를 쓰레기값으로 초기화 (초기화 안 하면 두 번째 실행 시, 첫 번째 실행할 때 저장한 요소가 영향을 끼침)

exp[i] = -100;

}

printf("\n원하는 계산을 입력해주세요 ");

printf("( '='은 제외하고 입력해주세요!! )\n:");

scanf\_s("%s", exp, max);

}

char menu() {

num = 0; num2 = 0; // get\_symbol에서 필요한 변수 초기화

printf("--------------- SH Calculator ---------------\n\n");

printf("( +,-,\*,/,^,나머지 연산이 가능합니다!! )\n");

printf("한 번만 계산을 하실거면 '1'을, 아니면 아무 숫자를 입력해주세요!!:");

scanf\_s("%d",&stop);

return stop;

}

int main() {

char exp[max]; // 중위식이 담길 배열

int pexp[max]; // 후위식이 담길 배열 (피연산자는 그냥 피연산자로, 연산자는 '-(아스키코드)'로 바뀌어서 이 배열에 담김)

// '-(아스키코드)'로 하는 이유는, 프로그램에서 아스키코드를 일반 수와 착각할 수 있기 때문에!!

do {

init(&st);

stop = menu();

get\_exp(exp); // 사용자로부터 중위식(원래식)을 입력받음

postfix(exp, pexp); // 중위식-> 후위식 변환

int res = eval(pexp); // 수식 계산

display(exp, pexp, res); // 결과를 출력

} while (stop != 1);

}

◆ C언어로 코딩하기

스택을 이용하지 않고 계산기를 만든 적이 있습니다. 당시 후위식이라는 개념도 모르고 알고리즘도 제대로 몰라서 헤맨 기억이 납니다.

스택을 이용한 후위식 변환, 검사는 완전한 알고리즘을 요구했고 그렇기에 오류없는 계산기를 만들 수 있었습니다. 이에 만족감을 느낍니다.

◇ 레포트 작성 후기