

Data structure

Project :: 전자계산기

과목 : 데이터 구조
담당교수 : 김인겸 교수님
발표자 : 20190895 김찬영
발표일시 : 2022.05.11

목차

01 과제 목표 분석

02 스택을 이용한 전자계산기 구현

03 스택 & 큐

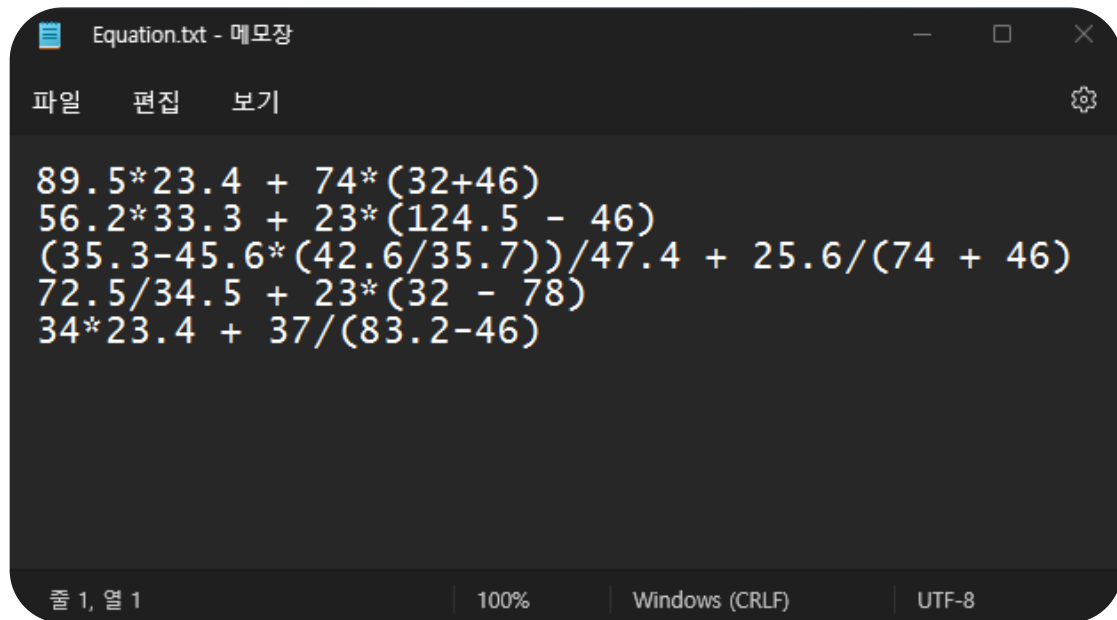
04 main() 분석

- 1) 여러 수식이 저장된 Equation.txt 파일
- 2) Main()함수 Argument를 입력받음
- 3) Check값에 따라 0일 때 큐, 1일 때 스택을 이용해 수식을 저장
- 4) 순서대로 수식을 꺼내 계산 후 결과값을 출력!

01

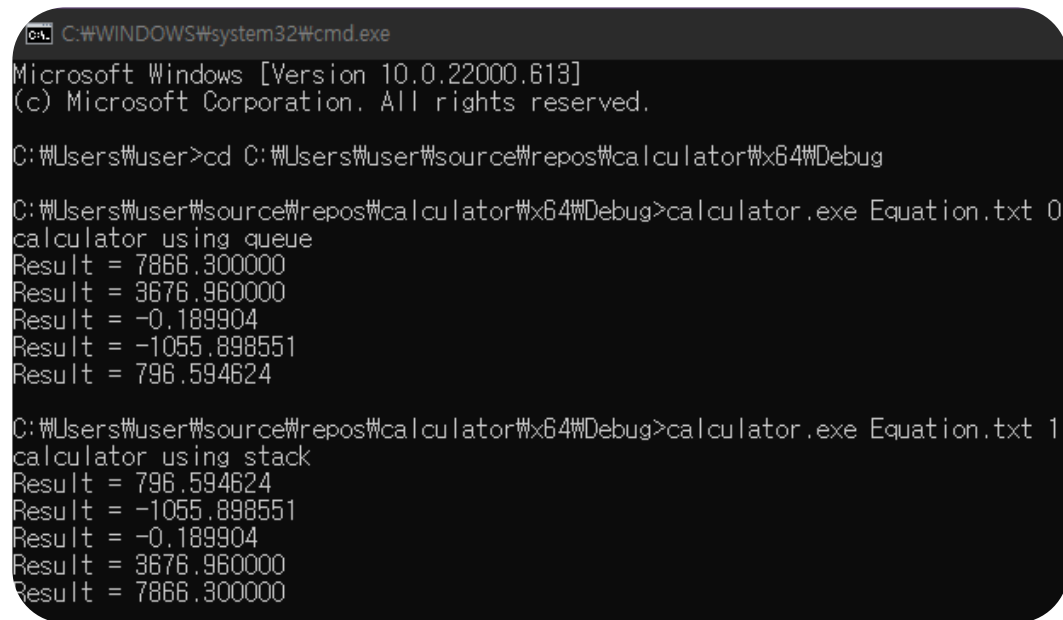
과제 목표 분석

어떠한 프로그램을 만들어야 하는가?



```
Equation.txt - 메모장
파일 편집 보기
89.5*23.4 + 74*(32+46)
56.2*33.3 + 23*(124.5 - 46)
(35.3-45.6*(42.6/35.7))/47.4 + 25.6/(74 + 46)
72.5/34.5 + 23*(32 - 78)
34*23.4 + 37/(83.2-46)
줄 1, 열 1 100% Windows (CRLF) UTF-8
```

(1) Equation.txt 안의 수식



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>cd C:\Users\User\source\repos\calculator\x64\Debug

C:\Users\User\source\repos\calculator\x64\Debug>calculator.exe Equation.txt 0
calculator using queue
Result = 7866.300000
Result = 3676.960000
Result = -0.189904
Result = -1055.898551
Result = 796.594624

C:\Users\User\source\repos\calculator\x64\Debug>calculator.exe Equation.txt 1
calculator using stack
Result = 796.594624
Result = -1055.898551
Result = -0.189904
Result = 3676.960000
Result = 7866.300000
```

(2) 결과 화면

이전의 과제에서 프로그래밍한 전자계산기와 다른 점?

- 1) 한 자리의 수가 아닌, 두 자리 이상의 실수를 처리해야 한다
- 2) Txt 파일로부터 수식을 읽어야 한다
- 3) 스택과 큐를 사용해 수식을 저장했다가 계산해야 한다

02 스택을 이용한 전자계산기 구현 알고리즘

괄호 검사

`check_matching()`

중위 표기식 -> 후위 표기식

`infix_to_postfix()`

후위 표기식을 계산

`calc_postfix()`

$32.14 - 5 * 4.1$

각각의 실수와 연산자를 구분하는 방법?

How to $(32.14 - 5 * 4.1) \Rightarrow (32.14 \ 5 \ 4.1 \ * \ -) ?$

- 1) 수식을 한 문자씩 읽음
- 2) 만약 숫자일 경우, 숫자 및 점이 끝날 때 까지 배열에 저장
- 3) 만약 괄호나 연산자일 경우, 우선순위를 판단한다
- 4) 각각 실수와 연산자 사이에는 공백을 추가한다

02

스택을 이용한 전자계산기 구현

infix_to_postfix()

```

/*중위 표기 수식을 후위 표기 수식으로 변환하는 함수
한 자리의 숫자가 아니기 때문에 실수와 실수, 실수와 연산자를 구분할 수 있도록 각각의 사이에 공백을 추가해 구분한다.*/

void infix_to_postfix(char* expr, char* postexpr) {
    init_stack();
    while (*expr != '\0') { //문자열의 끝을 만나기 전 까지 반복

        if (*expr >= '0' && *expr <= '9') { //조건문 : 실수를 읽기 위한 조건문. 숫자나 점을 만났을 경우
            do {
                *postexpr++ = *expr++; //postexpr에 expr의 값을 대입하고 각각 포인터가 가리키는 위치 1 증가하면 다음 글자를 계속 읽을 수 있다
            } while ((*expr >= '0' && *expr <= '9') || *expr == '.'); //do-while문을 이용해 계속 postexpr에 대입한다
            *postexpr++ = ' '; //실수를 다 읽으면 postexpr에 공백을 추가
        }

        else if (*expr == '(') { //조건문 : 좌괄호를 만났을 경우
            push(*expr++); //우선 스택에 저장하고 expr의 위치 증가
        }

        else if (*expr == ')') { //조건문 : 우괄호를 만났을 경우
            while (peek() != '(') { //좌괄호를 만나기 전 까지(반복문 안에서 pop을 하기 때문에 peek을 사용해도 무한 반복되지 않는다)
                *postexpr++ = pop(); //스택에 저장된 연산자를 pop 해서 postexpr에 대입한다
                *postexpr++ = ' '; //연산자를 pop 하면서 연산자 사이에 공백을 추가
            }
            pop(); //좌괄호를 스택에서 pop한다
            expr++; //괄호 안의 연산자를 다 꺼냈을 경우 expr 위치 증가
        }
    }
}

```

```
else if (*expr == '+' || *expr == '-' || *expr == '*' || *expr == '/') { //조건문 : 연산자를 만났을 경우
    while (!is_empty() && (precedence(*expr) <= precedence(peek()))) { //스택이 비어있지 않고, expr의 연산자가 스택에 저장된 연산자보다 우선순위가 낮을 경우
        *postexpr++ = pop(); //스택에 저장되어있는 연산자가 우선순위가 높으니 pop 해서 postexpr에 대입
        *postexpr++ = ' '; //각 연산자 사이에 공백을 추가
    }
    push(*expr); //위 문장이 끝나면 expr의 연산자를 스택에 추가한다.
    expr++; //모든 작업이 끝나고 expr 1 증가
}

else expr++; //실수, 괄호, 연산자에 포함이 되지 않을 경우 무시하고 expr 위치 1 증가시켜 다음 문자를 읽는다.
}

while (!is_empty()) { //마지막에는 스택에 남아있는 연산자들을 pop하여 postexpr에 추가하면 끝
    *postexpr++ = pop();
    *postexpr++ = ' '; //각 연산자 사이에 공백 추가
}

postexpr--;
*postexpr == '\0'; //마지막 연산자까지 모두 출력하게 되면 수식의 끝에 공백이 들어가게 되는데, 이를 NULL로 바꿔준다
}
```

02 스택을 이용한 전자계산기 구현

infix_to_postfix()

```
Equation.txt - 메모장
파일 편집 보기
89.5*23.4 + 74*(32+46)
56.2*33.3 + 23*(124.5 - 46)
(35.3-45.6*(42.6/35.7))/47.4 + 25.6/(74 + 46)
72.5/34.5 + 23*(32 - 78)
34*23.4 + 37/(83.2-46)
줄 1, 열 1 100% Windows (CRLF) UTF-8
```

```
C:\Users\user\source\repos\calculator\x64\Debug>ca
89.5 23.4 * 74 32 46 + * +
56.2 33.3 * 23 124.5 46 - * +
35.3 45.6 42.6 35.7 / * - 47.4 / 25.6 74 46 + / +
72.5 34.5 / 23 32 78 - * +
34 23.4 * 37 83.2 46 - / +
```

Infix_to_postfix() 함수 결과 화면

02 스택을 이용한 전자계산기 구현

calc_postfix()

32.14 5 4.1 * -

공백으로 각 실수의 구분은 했는데...
배열에서 실수로 바꿔야 계산이 가능하다

02 스택을 이용한 전자계산기 구현

calc_postfix()

32.14 5 4.1 * -

공백을 이용하여 실수를 구분했는데, 연산자 사이에도 공백이 존재

공백을 읽기 전에 읽었던 문자가 숫자일 때?

=> 그것은 실수가 끝났음을 알려주는 공백이다

02 스택을 이용한 전자계산기 구현

calc_postfix()

atof(*char str) : 문자열을 double형 변수로 바꿔준다

memset(*char str, value, size) : 배열을 특정 값으로 초기화

(1) buffer = { 0 , 0 , 0 , ... , 0 };

(2) 실수 읽어서 buffer에 대입 { 4 , 3 , . , 1 , 2 , 3 , ... , 0 }

(3) val = atof(buffer); 실행시 val = 43.123000

(4) memset(buffer, '0', strlen(buffer)*sizeof(char));
버퍼를 size만큼 '0'으로 초기화

02 스택을 이용한 전자계산기 구현

calc_postfix()

```
/*후위 표기 수식 계산 함수*/
double calc_postfix(char expr[]) {
    double val, val1, val2;
    int i = 0; //expr을 스캔하기 위한 변수
    int j = 0; //buffer에 문자를 입력하기 위한 변수
    char buffer[100] = { '0', }; //실수를 입력받을 buffer, 모든 문자를 NULL이 아닌 '0'으로 초기화
    char c; //expr을 한 문자씩 스캔할 때 사용하는 변수

    init_stack();
    while (expr[i] != '\0') { //expr의 끝을 만나기 전 까지 반복
        c = expr[i++]; //우선 한 글자씩 c에 대입한다

        if ((c >= '0' && c <= '9') || c == '.') { buffer[j++] = c; } //만약 c가 0과 9 사이의 숫자이거나 점일 경우 우선 버퍼에 대입한다
        else if (c == ' ' && (expr[i - 2] >= '0' && expr[i - 2] <= '9')) { //만약 c가 공백이며 그 이전의 문자가 숫자로 끝났을 경우는? 한 실수의 스캔이 끝났다는것을 의미한다
            val = atof(buffer); //버퍼의 내용을 atof()함수를 이용해 double형 변수로 val에 대입
            push(val); //val을 스택에 삽입
            j = 0;
            memset(buffer, 0, strlen(buffer) * sizeof(char)); //버퍼를 새로 사용하기 위해 j를 0으로 바꾸고 memset()함수로 버퍼의 모든 내용을 0으로 초기화
        }
    }
}
```

02 스택을 이용한 전자계산기 구현

calc_postfix().

```
else if (c == '+' || c == '-' || c == '*' || c == '/') { //연산자를 만났을 경우 pop 2회 실행해서 계산한 값을 스택에 저장
    val2 = pop(); //먼저 스택에서 나온 값이 뒤로 가야한다
    val1 = pop();
    switch (c) {
        case '+':push(val1 + val2); break;
        case '-':push(val1 - val2); break;
        case '*':push(val1 * val2); break;
        case '/':push(val1 / val2); break;
    }
}
else continue; //위 조건을 만족하지 않으면 반복문을 계속하여 다음 문자를 읽음
}
return pop(); //모든 반복이 끝나면 스택에는 최종 계산 결과가 남아있는데, 이를 pop해서 계산 결과를 반환
}
```


1) 전자계산기에 사용할 스택

2) 수식을 입력받을 스택

3) 수식을 입력받을 큐

*스택과 큐의 구현은 연결리스트로 구현

```
/*문자열을 저장하기 위해 선언한 연결리스트를 이용한 스택2
  기존의 스택 연산과 동일하며 노드 안에 문자열을 저장할 수 있게 하였다*/
typedef struct ListNode2 {
    char* str;
    struct ListNode2* link2;
}Node2;
Node2* top2 = NULL;
int is_emptystr() { return top2 == NULL; }
void init_stackstr() { top2 = NULL; }
void pushstr(char* expr) {
    Node2* p = (Node2*)malloc(sizeof(Node2));
    p->str = expr;
    p->link2 = top2;
    top2 = p;
}
char* popstr() {
    Node2* p;
    char* str;
    if (is_emptystr())
        error("스택 공백 에러");
    p = top2;
    top2 = p->link2;
    str = p->str;
    free(p);
    return str;
}
```

03

스택 & 큐

Check==0 일 때.

```

//연결리스트를 이용한 큐의 구현
typedef char* Element_queue;
typedef struct LinkedNode_queue {
    Element_queue data;
    struct LinkedNode_queue* link3;
} Node_queue;
//front와 rear의 선언
Node_queue* front = NULL, * rear = NULL;
//큐가 비어있는지 확인하는 함수
int is_empty_queue() { return front == NULL; }
//큐를 초기화하는 함수
void init_queue() { front = rear = NULL; }
//큐 안의 요소의 개수를 반환하는 함수
int size_queue() {
    Node_queue* p;
    int count = 0;
    for (p = front; p != NULL; p = p->link3) count++;
    return count;
}

```

```

//큐 삽입 연산
void enqueue(Element_queue e) {
    Node_queue* p = (Node_queue*)malloc(sizeof(Node_queue));
    p->data = e;
    p->link3 = NULL;

    if (is_empty_queue()) front = rear = p;
    else {
        rear->link3 = p;
        rear = p;
    }
}

//큐 삭제 연산
Element_queue dequeue() {
    Node_queue* p;
    Element_queue e;
    if (is_empty_queue())
        error("큐 공백 오류");
    p = front;
    front = front->link3;
    if (front == NULL) rear = NULL;
    e = p->data;
    free(p);
    return e;
}

//큐 peek 연산
Element_queue peek_queue() {
    if (is_empty_queue())
        error("큐 공백 오류");
    return front->data;
}

```

04 main() 분석

```
//main함수 실행. argument를 입력받아 argv[2]=0이면 큐를 사용해 수식을 저장, argv[2]=1이면 스택을 사용해 수식을 저장
int main(int argc, char* argv[]) {
    FILE* fp = NULL;    //파일 포인터 선언
    int check; //argv[2] 를 저장하기 위한 변수
    int i;
    char** equation_f = NULL, ** input_equation = NULL; //입력받은 수식을 저장할 두 개의 char이중포인터 선언
    int count = 0; //문자열이 몇 줄인지 세기 위한 함수

    if (argc != 3) error("exec equation check(0 or 1)"); //argument count가 3이 아닐 경우는 명령어가 잘못 입력된 경우이다

    fp = fopen(argv[1], "rt"); //argument value의 두 번째로 입력받은 파일 경로를 텍스트 읽기 모드로 오픈
    check = atoi(argv[2]); //argument value의 세 번째로 입력받은 내용을 atoi()함수를 사용해 정수로 변환 후 check에 대입

    //아래는 이중 포인터를 동적할당하는 과정이다. 배열의 내용을 초기화하기 위해 calloc()함수를 사용하였다
    equation_f = (char**)calloc(sizeof(char*) * 10, sizeof(char*));
    for (i = 0; i < 10; i++)
        equation_f[i] = (char*)calloc(sizeof(char) * 256, sizeof(char));

    input_equation = (char**)calloc(sizeof(char*) * 10, sizeof(char*));
    for (i = 0; i < 10; i++)
        input_equation[i] = (char*)calloc(sizeof(char) * 256, sizeof(char));
```

04 main() 분석

```
//파일 경로가 잘못되었을 경우
if (fp == NULL)
    error("fopen is failed");

if (check == 0) {    //0번 옵션일 경우 큐를 이용한 계산기 실행
    printf("calculator using queue\n");

    for (i = 0; fgets(input_equation[i], 256, fp) != NULL; i++) {    //txt파일로부터 끝이 아닐 때 까지 문자열을 한 줄씩 입력받음
        if (check_matching(input_equation[i]) != 0) //가장 먼저 괄호검사를 실행한다
            error("괄호 검사 실패");
        enqueue(input_equation[i]); //순서대로 문자열을 큐에 저장하고, 몇 줄을 입력받았는지 count
        count++;
    }
    for (i = 0; i < count; i++) {
        infix_to_postfix(dequeue(), equation_f[i]); //dequeue_str()함수로 한 줄씩 꺼내 계산 후 결과를 출력한다.

        printf("Result = %lf\n", calc_postfix(equation_f[i]));
    }
}
```

04 main() 분석

```
else if (check == 1) { //1번 옵션일 경우 스택을 이용한 계산기 실행
    printf("calculator using stack\n");

    for (i = 0; fgets(input_equation[i], 256, fp) != NULL; i++) { //txt파일로부터 끝이 아닐 때 까지 문자열을 한 줄씩 입력받음
        if (check_matching(input_equation[i]) != 0) //가장 먼저 괄호검사를 실행한다
            error("괄호 검사 실패");
        pushstr(input_equation[i]); //순서대로 문자열을 스택에 저장하고, 몇 줄을 입력받았는지 count
        count++;
    }
    for (i = 0; i < count; i++) {
        infix_to_postfix(popstr(), equation_f[i]); //popstr()함수로 한 줄씩 꺼내 계산 후 결과를 출력한다.
        printf("Result = %lf\n", calc_postfix(equation_f[i]));
    }
}

//할당된 메모리를 모두 해제하고 열었던 파일을 닫는다
for (i = 0; i < 10; i++)
    free(equation_f[i]);
free(equation_f);
for (i = 0; i < 10; i++)
    free(input_equation[i]);
free(input_equation);
fclose(fp);

return 0;
}
```

04 main() 분석

```
calculator using queue  
Result = 7866.300000  
Result = 3676.960000  
Result = -0.189904  
Result = -1055.898551  
Result = 796.594624
```

```
calculator using stack  
Result = 796.594624  
Result = -1055.898551  
Result = -0.189904  
Result = 3676.960000  
Result = 7866.300000
```

큐 : 선입선출 방식
스택 : 후입선출 방식

감사합니다

과목 : 데이터 구조
담당교수 : 김인겸 교수님
발표자 : 20190895 김찬영
발표일시 : 2022.05.11