Infinity Calculator

| 시스템 프로그래밍 기초 | 제출일자: 2018 년 12 월 3 일 | 담당교수: 강경태

| **2018044320 김규진**
| **2018044375 김상준**
| **2018044420 김영민**
| **2018044439 김영웅**
| **2018044302 권태현**

目 次

I. 개요

시스템 프로그래밍 수업에서 배운 파일에서 입력 값 받아오기, 스택, 연결리스트 등과 실습시간에 배운 실습과제 10 주차까지를 이용하여 무한 수 계산기 만들기 프로젝트를 시작하였다. 구체적으로는 무한 자리 수를 받기 위해서 이중연결리스트(Double Linked List 이하 DLL)를 이용하여 처리하는 방법을 강구하고 또한 중위표기로 입력된 식을 후위표기로 변환하는 과정에서 스택의 이용을 구상하였으며 시스템 프로그래밍 수업에서 학습한 파일에서 입력을 받아오는 방법 그리고 여러 가지의 예외에 대하여 올바른 입력을 받을 수 있도록 처리를 하였다. 이외에도 최적화 관련해서 토론을 하고 그 방법에 대하여 논하였다.

II. 입력 값 예외 처리

일반적으로는 정상적으로 연산식이 입력되겠지만, 항상 입력으로 들어오는 연산식이 정상적인 형식으로 들어오는 것이 아니다. 사람이 보기에는 올바르지 못한 입력이 컴퓨터에게 있어서는 입력이 잘못된 것인지 판단하지 못하고 실행이 되고 오류가 발생하게 된다. 따라서 우리는 비정상적인 연산식이 들어오는 경우를 세부적으로 나눠서 하나의 예외도 빠지지 않도록 예외경우를 제거해야 했다.

----1) 예외처리

우리는 코드에서 연산을 입력 받을 때에 Operator(연산자)가 연속적으로 입력이 될 때의 경우, 열린 괄호와 닫힌 괄호의 수가 일대일 대응되지 않을 경우, 연산식이 Operator(연산자)로 끝났을 경우, '.'뒤(소수점 뒤)에 숫자가 없을 경우, 하나의 Operand(피연산자)에 '.'이(소수점) 두 개 이상으로 올 경우등과 같이 많은 예외들이 있을 수 있으므로 이를 처리해주었다.

----2) 구현설명

```
1  void getnumber(DLL *list, FILE *ifp) {
2  int count = 0;
3  char temp;
4  while (1) {
5  temp = getc(ifp);
6  if (temp == '₩n') break;
7  if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-
8  ' || temp == '*' || temp == '/') {
9  if (count == 0) {
10 if (temp == '+') {
11 temp = getc(ifp);
12 count++;
13 }
14 if (temp == '-') {
15 list->swh = 2;
16 temp = getc(ifp);
17 count++;
```

파일에서 입력 값을 입력 받는 함수에서 '-3 + 5'와 같은 예외를 처리

```
18 }
19 }
20 if (temp == '+' || temp == '-' || temp == '*') list->i = list->i + 1;
21 append(list, newnode(temp));
22 count++;
23 ///////////////
24 temp = getc(ifp);
25 if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-
26 ' || temp == '*' || temp == '/') {
27 printf("ERROR: Worng Input !");
28 break;
29 }
30 else if (isblank(temp)) count++;
31 else if (isalpha(temp)) {
32 printf(" Wrong Input ! ");
33 break;
34 }
35 else if (isdigit(temp)) {
36 append(list, newnode(temp));
```

24 라인: 연산자의 연속

30 라인: 알파벳이 입력

숫자 이외 값 입력을 처리 .

```
37 count++;
38 }
39 else {
40 printf(" Wrong Input ! ");
41 break;
42 }
43 //////////////
44 }
45 else if (isblank(temp)) count++;
46 else if (isalpha(temp)) {
47 printf(" Wrong Input ! ");
48 break;
49 }
50 else if (isdigit(temp)) {
51 append(list, newnode(temp));
52 count++;
53 }
54 else {
55 printf(" Wrong Input ! ");
```

```
56  break;
57  }
58  }
    }
```


Colored by Color Scripter

III. 문자열 길이 제한

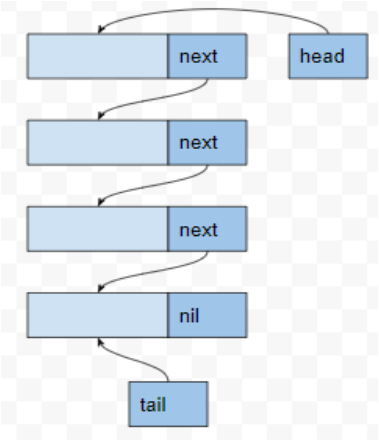
프로젝트에서 원하는 아주 큰 수를 처리하기 위해서 C 언어에서 제공하는 기본 자료형에는 한계가 있다.

자료형	크기	범위	비고
char signed char	1바이트, 8비트	-128~127	
unsigned char	1바이트, 8비트	0~255	
short short int	2바이트, 16비트	-32,768~32,767	int 생략 가능
unsigned short unsigned short int	2바이트, 16비트	0~65,535	int 생략 가능
int signed int	4바이트, 32비트	-2,147,483,648~ 2,147,483,647	
unsigned unsigned int	4바이트, 32비트	0~4,294,967,295	int 생략 가능
long long int signed long signed long int	4바이트, 32비트	-2,147,483,648~ 2,147,483,647	int 생략 가능
unsigned long unsigned long int	4바이트, 32비트	0~4,294,967,295	int 생략 가능
long long long long int signed long long signed long long int	8바이트, 64비트	-9,223,372,036,854,775,808~ 9,223,372,036,854,775,807	int 생략 가능
unsigned long long unsigned long long int	8바이트, 64비트	0~18,446,744,073,709,551,615	int 생략 가능

위의 표를 참고하면, unsigned long long 이 가장 큰 수를 표현할 수 있는데 물론 매우 큰 수이지만 무한 수 계산기에서 제공하려는 수에 비해서는 턱없이 작다고 할 수 있다.

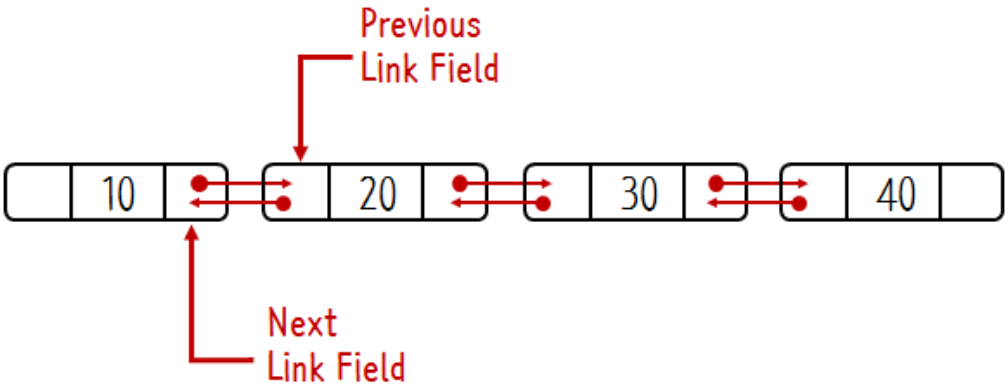
따라서 우리는 이미 정해져 있는 자료형에 맞춰서 값을 배정(assignment)하는 방식이 아니라 입력 값을 받을 때에 각각의 수를 한 자리씩 이중 연결 리스트(DLL)을 이용하여 저장하는 동적 메모리 할당방법을 선택했다.

----1) DLL



연결리스트(linked list)란, 각 Node(이하 노드)가 데이터와 포인터를 가지고 한 줄로 연결되어 있는 방식으로 데이터를 저장하는 자료 구조의 하나이다. 각 노드 당 포인터가 하나만 있어서 단방향성이므로 데이터의 삭제와 데이터의 추가 같은 수정을 하기 어렵다. 예를 들어 1234 라는 수를 각 자리에 저장했다고 하면 1 에서 다음에 있는 노드인 2 값으로 이동하기 는 next 를 통해 쉽지만 반대로 2 에서 1 노드로는 이동하기가 어렵다는 점이다.

반면에 이중연결리스트(Doubly linked list)는 연결리스트와 다르게 노드가 이전 노드(prev)포인터와 다음 노드 (next)포인터로 구성되어 있다.



DLL 은 양방향으로 연결되어 있기 때문에 다음 노드로 이동하는 것과 마찬가지로 이전 노드를 prev 포인터를 이용하여 탐색할 수 있어 양방향성이라는 것이 장점이다.

이는 데이터의 수정 및 접근에 있어서 편리성을 제공해주지만 각 값을 저장할 때 마다 연결리스트에 비해서 이전(prev) 포인터 하나의 값을 더 저장해줘야 하므로 메모리 측면에 있어서는 단점이 존재한다.

----2) DLL 구현설명

```
1  typedef struct Node {
2  char val;
3  struct Node *next;
4  struct Node *prev;
5  } Node;
6
7  typedef struct {
8  Node *head;
9  int size;
10 int size_1;
11 int size_2;
12 int swh;
13 int i;
14 } DLL;
15
16 void append(DLL *list, Node *newnode) {
17 Node *curr;
```

뒤에 쓰이는 변수들이지만

size_1 은 소수점 자릿수를 저장하는 변수

size_2 은 자연수 자리수를 저장하는 변수

swh 은 stack_3 의 부호표시를 위한 switch

i 연산자의 개수의 변수이다.

```
18 if (list->head == NULL) {
19 list->head = newnode;
20 }
21 else {
22 curr = list->head;
23 while (curr->next != NULL) {
24 curr = curr->next;
25 }
26 newnode->prev = curr;
27 curr->next = newnode;
28 }
29 list->size = list->size + 1;
30 }
31 void getnumber(DLL *list) {
32 int count = 0;
33 while (1) {
34 char temp = getc (ifp);
35 if (temp == '₩n') break;
36
```

```
37 else if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-
' || temp == '*' || temp == '/') {
38 if (count == 0) {
39 if (temp == '+') {
40 temp = getc (ifp);
41 count++;
42 }
43 if (temp == '-') {
44 list->swh = 2;
45 temp = getc (ifp);
46 count++;
47 }
48 }
49 if (temp == '+' || temp == '-' || temp == '*') list->i = list->i + 1;
50 append(list, newnode(temp));
51 count++;
52 ///////////////
53 temp = getc (ifp);
54
55
```

```
56 if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-
' || temp == '*' || temp == '/') {
57 printf("ERROR: Worng Input !");
58 break;
59 }
60 else if (isblank(temp)) count++;
61 else if (isalpha(temp)) {
62 printf(" Wrong Input ! ");
63 break;
64 }
65 else if (isdigit(temp)) {
66 append(list, newnode(temp));
67 count++;
68 }
69 else {
70 printf(" Wrong Input ! ");
71 break;
72 }
73 }
74
```

```
75 else if (isblank(temp)) count++;
76 else if (isalpha(temp)) {
77 printf(" Wrong Input ! ");
78 break;
79 }
80 else if (isdigit(temp)) {
81 append(list, newnode(temp));
82 count++;
83 }
84 else {
85 printf(" Wrong Input ! ");
86 break;
87 }
88 }
   }
```

Colored by Color Scripter

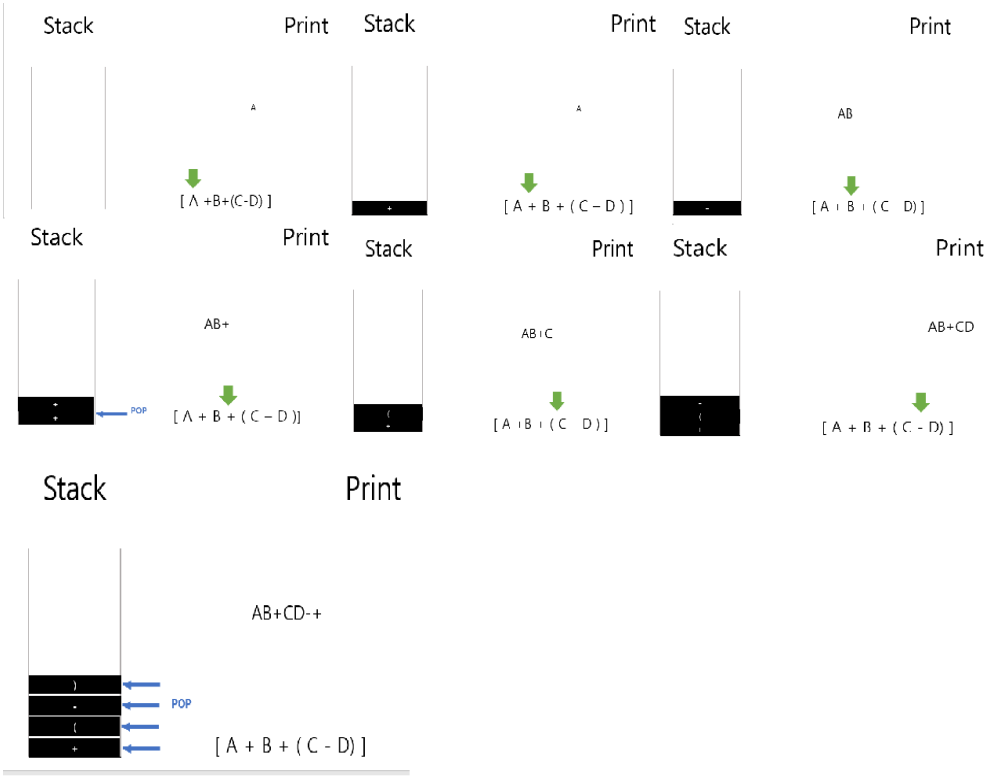
IV. 수식 표기법 변환

사람과 다르게 컴퓨터는 연산과정을 순차적으로 처리한다. 컴퓨터의 이러한 특성으로 인해 사람이 사용하는 중위표기법으로는 컴퓨터의 연산에 어려움이 있다. 하지만 후위표기법을 사용한 식으로 계산할 때에는 연산자의 우선순위와 괄호로 인한 직관적이지 못한 상황을 방지하는 장점이 있다. 이러한 이유로 인해 컴퓨터가 연산 할 때에는 후위 표기법을 사용한다.

----1) 후위 표기법(postfix notation)

컴퓨터에서 연산을 보다 쉽게 하기 위해서 연산자(Operator)를 피연산자(Operand)보다 뒤에 표기하는 것을 후위 표기법이라 한다.

예를 들어서 A+B+(C-D)와 같은 연산식을 후위 표기법으로 변환하면 아래의 이러한 과정을 거쳐 중위를 후위표기법으로 변환한다.



----2) 중위 → 후위 구현

```
1  int GreaterOpr(char opr1, char opr2)
2  {
3  if (opr1 == '*') {
4  if (opr2 == '+' || opr2 == '-' || opr2 == ')')
5  return TRUE;
6  else if (opr2 == '*')
7  return TRUE;
8  else if (opr2 == '(')
9  return FALSE;
10 }
11 else if (opr1 == '+' || opr1 == '-') {
12 if (opr2 == '+' || opr2 == '-' || opr2 == ')')
13 return TRUE;
14 else if (opr2 == '*' || opr2 == '(')
15 return FALSE;
16 }
17 else if (opr1 == '(') {
18 if (opr2 == ')')
```

연산자의 우선순위를 결정하는 함수

opr1 이 opr2 보다 우선순위가 높으면 TRUE 를 return

TRUE 와 FALSE 는 #define 을 통해 1 과 0 으로 정의

```
19 return TRUE;
20 else
21 return FALSE;
22 }else if (opr1 == ')')
23 return TRUE;
24 }
25 Colored by Color Scripter
```

괄호를 포함한 연산자
우선순위에 따라 프로그래밍

```
1  void POP_all(DLL *stack, DLL *list_1)
2  {
3  Node *cur = stack->head;
4  while (cur->next != NULL) {
5  cur = cur->next;
6  }
7  while (1) {
8  if (cur->prev == NULL) {
9  append(list_1, newnode(cur->val));
10 break;
```

더 이상 append 할 Node 가 없을 때 스택의 남아있는 연산자를 모두 Pop 해주는 함수

```
11 }
12 else {
13 append(list_1, newnode(cur->val));
14 cur = cur->prev;
15 }
16 }
17 free(stack);
18 stack = newDLL();
19 }
```

Colored by Color Scripter

list_1:후위표기법 변환 리스트

stack: 연산자 보관 리스트

```
1  void PushOrPop(DLL *stack, char input_opr, DLL *list_1)
2  { //Push 할건지 Pop 할건지 판단하는 함수
3  while (1) {
4  int ssize = stack->size;
5  if (ssize == 0) {
6  append(stack, newnode(input_opr));
7  ssize += 1;
8  break;
```

GreatOpr 함수를 이용

연산자를 어느 스택에 Push 나 Pop 을 할 것 인지 판단하는 함수

```
9  }
10 else {
11 Node *bigyo = stack->head;
12 while (bigyo->next != NULL)
13 bigyo = bigyo->next;
14 if (GreaterOpr(bigyo->val, input_opr) == FALSE) {
15 append(stack, newnode(input_opr));
16 ssize += 1;
17 break;
18 }
19 else {
20 if (ssize == 1) {
21 POP_all(stack, list_1);
22 append(stack, newnode(input_opr));
23 ssize += 1;
24 break;
25 }
26 else if (ssize >= 2) {
27 append(list_1, newnode(bigyo->val));
```

```
28 bigyo = bigyo->prev;
29 bigyo->next = NULL;
30 ssize -= 1;
31 }
32 }
33 }
34 }
35 }
```

Colored by Color Scripter

```
1 void postfix(DLL *list, DLL *list_1) {
2 DLL *stack = newDLL();
3 Node *curr = list->head;
4 while (1) {
5 if (curr->next == NULL) {
6 append(list_1, newnode(curr->val));
7 append(list_1, newnode(' '));
8 break;
```

최종 후위표기 변환함수

```
9  }
10 else if (isdigit(curr->val) || curr->val == '.')
11 {
12 append(list_1, newnode(curr->val));
13 if (curr->next == NULL) break;
14 curr = curr->next;
15 }
16 else {
17 append(list_1, newnode(' '));
18 PushOrPop(stack, curr->val, list_1);
19 curr = curr->next;
20 }
21 }
22 POP_all(stack, list_1);
   }
```

Colored by Color Scripter

```
1  void delete_par(DLL *list_1) {
2  Node *curr = list_1->head;
3  if (curr->val == ' ') deleteAt(list_1, 0);
4  while (1) {
5  if (curr->next == NULL) {
6  if (curr->val == '(' || curr->val == ')') {
7  curr->prev->next = NULL;
8  }
9  break;
10 }
11 if (curr->val == '(' || curr->val == ')') {
12 curr->prev->next = curr->next;
13 curr->next->prev = curr->prev;
14 }
15 curr = curr->next;
16 }
17
18 }
```

Colored by Color Scripter

V 덧셈 & 뺄셈 보조 함수

```
1 void delete_all(DLL *list1) {
2 int a = list1->size;
3 for (int i = 0; i < a; i++)
4 deleteAt(list1, 0);
5 }
```

후위 표기법 따라 연산하고
해당 Node 를 비우는 함수

```
1   int deleteAt(DLL *list, int index) {
2   Node *curr = list->head;
3   int count = 0;
4   if (list->size > 1) {
5   if (index == 0) {
6   list->head->next->prev = NULL;
7   list->head = list->head->next;
8   list->size = list->size - 1;
9   return 0;
```

Index 를 이용 빈 Node 를
삭제하는 함수.

```
10  }
11  while (count != index) {
12  count++;
13  curr = curr->next;
14  if (curr == NULL) {
15  printf("DELETE ERROR: Out of Bound.₩n");
16  return 0;
17  }
18  }
19  if (count == index) {
20  if (curr->next == NULL) {
21  curr->prev->next = NULL;
22  list->size = list->size - 1;
23  return 0;
24  }
25  curr->prev->next = curr->next;
26  curr->next->prev = curr->prev;
27  list->size = list->size - 1;
28  }
```

```
29  }
30  else {
31  list->size = list->size - 1;
32  list->head->prev = NULL;
33  list->head->next = NULL;
34  list->head = NULL;
35  }
36  }
37
38  int insertAt(DLL *stack_3, int index, Node *newnode) {
39  Node *curr = stack_3->head;
40  int count = 0;
41  if (index < 0) {
42  printf("INSERT ERROR: Out of Bound.₩n");
43  return 0;
44. }
    if (index == 0) {
    stack_3->head = newnode;
    curr->prev = newnode;
```

소수점 삽입을 위한 함수로
index 에 소수점 삽입

```
stack_3->head->next = curr;

return 0;

}

while (1) {

if (curr->next == NULL) {

count++;

break;

}

else {

curr = curr->next;

count++;

}

}

index = count - index;

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

curr = curr->prev;

}

newnode->prev = curr;

newnode->next = curr->next;
```

```
curr->next->prev = newnode;

curr->next = newnode;

}
```

Colored by Color Scripter

```
1  void insert(DLL *list_1, DLL *list_3) {
2  append(list_3, newnode(' '));
3  Node *curr = list_1->head;
4  Node *curr_1 = list_3->head;
5  while (1) {
6  if (curr->val == '+' || curr->val == '-' || curr->val == '*') {
7  curr = curr->next;
8  break;
9  }
10 else {
11 curr = curr->next;
12 }
13 }
```

계산된 DLL 을 후위표기법 연산 식에 병합

```
14  while (1) {
15  if (curr_1->next == NULL) break;
16  else curr_1 = curr_1->next;
17  }
18  while (1) {
19  if (curr->next == NULL) {
20  append(list_3, newnode(curr->val));
21  break;
22  }
23  append(list_3, newnode(curr->val));
24  curr = curr->next;
25  }
26  }
27  int insertAt_int(DLL *list, int index, Node *newnode) {
28  int count = 0;
29  Node *curr = list->head;
30  if (index == 0) {
31  list->head = newnode;
32  curr->prev = newnode;
```

```
33  list->head->next = curr;
34  return 0;
35  }
36  while (count != index) {
37  count++;
38  curr = curr->next;
39  if (curr->next == NULL) break;
40  }
41  if (curr->next == NULL) {
42  newnode->prev = curr;
43  curr->next = newnode;
44  return 0;
45  }
46  newnode->prev = curr->prev;
    newnode->next = curr;
49  curr->prev->next = newnode;
50  curr->prev = newnode;
51  }
```

Colored by Color Scripter

```
1  void plus_zero(DLL *list_3) {
2  Node *curr = list_3->head;
3  while (1) {
4  if (curr->val == '.') {
5  if (curr->prev == NULL) {
6  insertAt(list_3, 0, newnode('0'));
7  break;
8  }
9  if (curr->prev->val == '-') {
10 insertAt(list_3, 1, newnode('0'));
11 }
12 }
13 if (curr->next == NULL) break;
14 curr = curr->next;
15 }
16 }
```

예를 들어 '1 – 1.1'과 같은 연산 식에서 답이 '–.1'이 도출됨.

해결을 위해 0 을 추가해주는 코드

Colored by Color Scripter

```
1  void plus_change(DLL *list_1) { // 앞의수가 -일경우 뒤에있는 +를 -
2  로 바꿔주기 위한 함수
3  Node *curr = list_1->head;
4  if (curr->val == '-') {
5  while (1) {
6  if (curr->next == NULL) break;
7  curr = curr->next;
8  if (curr->val == '+') {
9  curr->val = '-';
10 list_1->swh = 2;
11 deleteAt(list_1, 0);
12 break;
13 }
14 else if (curr->val == '-') {
```

앞의 수가 음수 일 경우에 뒤에 있는
연산자 '-'를 '+'로 바꾸어 주는 함수

```
15 curr->val = '+';
16 deleteAt(list_1, 0);
17 list_1->swh = 3;
18 break;
19 }
20
21 }
22 }
23 }
```

Colored by Color Scripter

```
1  void reverse(DLL *list, DLL *list_1) { //
2  Node *curr = list->head;
3  while (1) {
4  if (curr->next == NULL) break;
5  else curr = curr->next;
6  }
7  while (1) {
```

DLL 노드의 앞뒤를 바꾸어 주는 함수

```
8  if (curr->prev == NULL) {
9  append(list_1, newnode(curr->val));
10 break;
11 }
12 else {
13 append(list_1, newnode(curr->val));
14 curr = curr->prev;
15 }
16 }
17 }
```

Colored by Color Scripter

```
1  void size_check(DLL *list_1) {
2  Node *curr = list_1->head;
3  list_1->size = 0;
4  while (1) {if (curr->next == NULL) {
5  list_1->size = list_1->size + 1;
6  break;
```

List_1 의 size 를
체크해주는 함수

```
}
else {
list_1->size = list_1->size + 1;
curr = curr->next;
}
}
}
```

Colored by Color Scripter

```
void zero(DLL *stack_1, DLL *stack_2) {
Node *curr = stack_1->head;
Node *curr_1 = stack_2->head;
DLL *stack_3 = newDLL();
int a = 0;
int b = 0;
int c = 0;
int d = 0;
while (1) {
```

피연산자들의 정수와 소수의 자릿수를 0 을 넣어 맞춰주는 함수

```
10  if (curr->val == '.') break;
11  else if (curr->next == NULL) {
12  c++;
13  break;
14  }
15  else {
16  c++;
17  curr = curr->next;
18  }
19  }
20  while (1) {
21  if (curr->next == NULL) break;
22  else {
23  a++;
24  curr = curr->next;
25  }
26  }
27  while (1) {
28  if (curr_1->val == '.') break;
```

```
29  else if (curr_1->next == NULL) {
30  d++;
31  break;
32  }
33  else {
34  d++;
35  curr_1 = curr_1->next;
36  }
37  }
38  while (1) {
39  if (curr_1->next == NULL) break;
40  else {
41  b++;
42  curr_1 = curr_1->next;
43  }
44  }
45  stack_1->size_2 = c;
46  stack_2->size_2 = d;
47  if (a == 0 && b != 0) {
```

```
48  append(stack_1, newnode('.'));
49  }
50  if (b == 0 && a != 0) append(stack_2, newnode('.'));
51  if (c > d) {
52  int f = c - d;
53  for (int i = 0; i < f; i++) {
54  insertAt_int(stack_2, 0, newnode('0'));
55  }
56  }
57  else if (d > c) {
58  int f = d - c;
59  for (int i = 0; i < f; i++) {
60  insertAt_int(stack_1, 0, newnode('0'));
61  }
62  }
63  else {
64  }
65  if (a > b) {
66  int c = a - b;
```

'c' 와 'd' 비교를 통해
정수부분 자릿수를 맞춤

```
67  for (int i = 0; i < c; i++) {
68  append(stack_2, newnode('0'));
69  }
70  stack_1->size_1 = a;
71  stack_2->size_1 = a;
72  }
73  else if (b > a) {
74  int c = b - a;
75  for (int i = 0; i < c; i++) {
76  append(stack_1, newnode('0'));
77  }
78  stack_1->size_1 = b;
79  stack_2->size_1 = b;
80  }
81  else {
82  stack_1->size_1 = a;
83  stack_1->size_1 = b;
84  }
85  if (c < d) {
```

```
86  copy_1(stack_3, stack_1);
87  delete_all(stack_1);
88  copy_1(stack_1, stack_2);
89  delete_all(stack_2);
90  copy_1(stack_2, stack_3);
91  delete_all(stack_3);
92  stack_1->swh = 1;
93  stack_2->swh = 1;
94  }
95  else if (c == d) {
96  Node *curr_2 = stack_1->head;
97  Node *curr_3 = stack_2->head;
98  while (1) {
99  if (curr_2->val == '.' && curr_3->val == '.') {
100 curr_2 = curr_2->next;
101 curr_3 = curr_3->next;
102 }
103 int m = curr_2->val - 48;
104 int n = curr_3->val - 48;
```

```
105 if (n > m) {
106 copy_1(stack_3, stack_1);
107 delete_all(stack_1);
108 copy_1(stack_1, stack_2);
109 delete_all(stack_2);
110 copy_1(stack_2, stack_3);
111 delete_all(stack_3);
112 stack_1->swh = 1;
113 stack_2->swh = 1;
114 break;
115 }
116 if (curr_2->next == NULL && curr_3->next == NULL) break;
117 curr_2 = curr_2->next;
118 curr_3 = curr_3->next;
119 }
120 }
121 }
122
```

Colored by Color Scripter

V. 덧셈 뺄셈 함수 구현

```
1  void cal(DLL *list, DLL *stack_3) {
2  DLL *stack_1 = newDLL(); // 숫자 1
3  DLL *stack_2 = newDLL(); // 숫자 2
4  Node *curr = list->head;
5  while (1) { // + - 만날 때 까지 노드 움직임
6  if (curr->val == '+' || curr->val == '-') {
7  curr = curr->prev;
8  curr = curr->prev;
9  break;
10 }
11 else {
12 curr = curr->next;
13 }
14 }
15 while (1) { //첫 번째 인자 수 맨 앞 빈칸
16 if (curr->prev == NULL || curr->val == ' ') {
17 break;
18 }
19 else {
20 curr = curr->prev;
21 }
```

List: 후위표기법으로 바뀐 식

stack_3: 연산의 값을 저장하는 리스트.

연산자를 만날 때까지 반복

피 연산자들을 ' ' 로 구분 후 저장

```
22  }
23  curr = curr->next;
24  while (1) { //첫 번째 수 저장
25  if (curr->val == ' ') {
26  curr = curr->prev;
27  break;
28  }
29  else {
30  append(stack_2, newnode(curr->val));
31  curr = curr->next;
32  }
33  }
34  while (1) { //첫 번째 수 넘어감
35  if (curr->val == ' ' || curr->prev == NULL) {
36  curr = curr->prev;
37  break;
38  }
39  else {
40  curr = curr->prev;
41  }
42  }
43  while (1) { //두 번째수 앞 빈칸까지
44  if (curr->val == ' ' || curr->prev == NULL) {
45  break;
```

```
46  }
47  else {
48  curr = curr->prev;
49  }
50  }
51  while (1) { //두번째 수 저장
52  if (curr->val == ' ') {
53  break;
54  }
55  else {
56  append(stack_1, newnode(curr->val));
57  curr = curr->next;
58  }
59  }
60
61  zero(stack_1, stack_2);
62  if (list->swh == 2) {
63  stack_1->swh = 2;
64  stack_2->swh = 2;
65  list->swh = 0;
66  }
67  if (list->swh == 1) {
68  stack_1->swh = 1;
69  stack_2->swh = 1;
```

```
70  list->swh = 0;
71  }
72  if (list->swh == 3) {
73  stack_1->swh = 3;
74  stack_2->swh = 3;
75  list->swh = 0;
76  }
77  while (1) {
78  if (curr->val == '+' || curr->val == '-' || curr->val == '*') {
79  break;
80  }
81  else {
82  curr = curr->next;
83  }
84  }
85  if (curr->val == '+') {
86  Node *curr_1 = stack_1->head;
87  Node *curr_2 = stack_2->head;
88  while (1) { // 노드 끝까지 보냄
89  if (curr_1->next == NULL) break;
90  else curr_1 = curr_1->next;
91  }
92  while (1) { // 노드 끝까지 보냄
93  if (curr_2->next == NULL) break;
```

```
94  else curr_2 = curr_2->next;
95  }
96  int count = 0;
97  int a = curr_1->val - 48;
98  int b = curr_2->val - 48;
99  int c;
100 while (1) { // + 계산시작
101 // 수가 끝났을 경우 0 으로 처리
102 if (curr_1->val == '.') {
103 curr_1 = curr_1->prev;
104 }
105 if (curr_2->val == '.') {
106 curr_2 = curr_2->prev;
107 }
108 a = curr_1->val - 48;
109 b = curr_2->val - 48;
110 c = a + b;
111 if (count == 1) {
112 c++;
113 count = 0;
114 if (c >= 10) {
115 c = c - 10;
116 count = 1;
117 }
```

count : 올림수

'a' & 'b' : char val 를 int val 로 변환

```
118 }
119 else {
120 if (c >= 10) {
121 c = c - 10;
122 count = 1;
123 }
124 }
125 if (curr_1 != NULL) curr_1 = curr_1->prev;
126 if (curr_2 != NULL) curr_2 = curr_2->prev;
127
128 c = c + 48;
129 append(stack_3, newnode(c));
130 if (curr_1 == NULL && curr_2 == NULL) {
131 if (count == 1) {
132 append(stack_3, newnode('1'));
133 count = 0;
134 }
135 break;
136 }
137 } // while 문
138 int d = stack_1->size_1;
139
140 if (d > 0) insertAt(stack_3, d, newnode('.'));
141 if (stack_1->swh == 1) stack_3->swh = stack_1->swh;
```

자리수가 올라간 경우

```
142 if (stack_1->swh == 3) append(stack_3, newnode('-'));// -
143 3 5 - 인경우
144 stack_1->swh = 0;
145 stack_2->swh = 0;
146 stack_3->swh = 0;
147 } // if +문에 걸림
148
149 //// -계산
150 else if (curr->val == '-') {
151 Node *curr_1 = stack_1->head;
152 Node *curr_2 = stack_2->head;
153 while (1) { // 노드 끝까지 보냄
154 if (curr_1->next == NULL) break;
155 else curr_1 = curr_1->next;
156 }
157 while (1) { // 노드 끝까지 보냄
158 if (curr_2->next == NULL) break;
159 else curr_2 = curr_2->next;
160 }
161 int count = 0;
162 int a = curr_1->val - 48;
163 int b = curr_2->val - 48;
164 int c;
165 while (1) {
```

```
166 if (curr_1->val == '.') {
167 curr_1 = curr_1->prev;
168 }
169 if (curr_2->val == '.') {
170 curr_2 = curr_2->prev;
171 }
172 a = curr_1->val - 48;
173 b = curr_2->val - 48;
174 c = a - b;
175 if (count == 1) {
176 c--;
177 count = 0;
178 }
179 if (c < 0) {
180 c = c + 10;
181 count = 1;
182 }
183 if (curr_1 != NULL) curr_1 = curr_1->prev;
184 if (curr_2 != NULL) curr_2 = curr_2->prev;
185 c = c + 48;
186 append(stack_3, newnode(c));
187 if (curr_1 == NULL && curr_2 == NULL) {
188 break;
189 }
```

```
190 }
191 int d = stack_1->size_1;
192 stack_3->swh = stack_1->swh;
193 if (d > 0) insertAt(stack_3, d, newnode('.'));
194 Node *curr_3 = stack_3->head;
195 while (1) {
196 if (curr_3->next == NULL) break;
197 else curr_3 = curr_3->next;
198 }
199 while (1) {
200 if (curr_3->val != '0') break;
201 else {
202 curr_3 = curr_3->prev;
203 curr_3->next = NULL;
204 }
205 }
206 ////// stack_3 의 값이 -일경우
207 if (stack_3->swh == 1) append(stack_3, newnode('-'));
208 stack_1->swh = 0;
209 stack_2->swh = 0;
210 stack_3->swh = 0;
211 }
    }
```

Colored by Color Scripter

자연수의 앞자리가 0 일 경우에 0 삭제

VI. 프로그램 최적화 및 발전방향

컴퓨터에서 최적화란 컴파일러에 소스프로그램을 생성하는 과정에 있어 실행시간을 단축하거나 메모리 영역을 최소화하는 것을 의미한다. 따라서 공통식이나 불필요한 변수정의를 제거하는 것, 루프 불변식을 루프 밖으로 이동시키는 작업을 말한다. 최적화의 종류에는 여러 가지가 있지만 그 중 첫 번째는 컴파일러 최적화이다.

컴파일러 최적화는 컴파일러에서 출력되는 프로그램의 효율성을 최적화하는 과정을 말한다. 일반적으로 실행 속도를 최대화하고 메모리의 양을 최소화하기 위해 많이 이용된다. 주로 최적화 옵션을 이용하여 컴파일러의 최적화를 실행시킨다.

사실 최적화를 하기 위해서 실행 속도를 최대로 하고 사용하는 메모리의 크기를 최소로 하는 것이 가장 이상적이다. 하지만 최적화는 만능일 수 없기 때문에 속도와 크기, 성능과 가독성 각각의 항목들의 등가교환을 통해서 이루어진다. 속도 측면에서는 배열을 사용하는 것이 효율적이지만 큰 메모리의 이동과 같은 실행을 효과적으로 하기 위해서는 포인터 사용이 효율적이다. 따라서 이번 프로젝트에서는 무한수라는 큰 메모리를 사용하므로 이를 위해 포인터를 사용하였다.

하지만 이러한 최적화를 통해서 만들어진 코드가 실행되었을 때 컴파일러의 잘못된 판단에 의해서 프로그램이 의도하지 않는 동작을 하거나 필요한 동작이 생략될 수도 있다. 이를 해결하는 방법은 volatile 이다. 프로그램이 실행될 때 속도를 위한 데이터를 캐시로부터 읽어온다. 하지만 하드웨어에 의해서 변경된 값들은 주 메모리에서 직접 읽어오도록 해야 한다. 이러한 특성을 이용하기 위해 하드웨어가 사용하는 메모리를 volatile 로 선언해야 한다. 그렇게 volatile 로 선언된 변수는 최적화 후에도 사라지지 않고 남아 최적화가 잘못되는 경우를 막아준다.

이번 과제에서 후위표기의 연산식의 괄호의 처리를 완벽하게는 구현하지 못하였다. 덧셈과 뺄셈에서는 괄호가 연산결과에 영향이 없는 반면, 곱셈이 들어간 연산에서는 문제가 될 여지가 충분하므로 다음 과제에서 곱셈을 포함한 완벽한 괄호 계산을 목표로 할 것이다. 하지만 괄호 계산을 제외한 모든 계산이 원할하게 실행되는 것을 알 수 있었다.

또한 팀 프로젝트인 점을 고려하여 후에 코드의 수정, 삭제, 추가의 용이성을 위하여 각 함수 및 변수의 이름을 수정하기로 하였다. 최적화 측면에서는 메모리의 사용 증가를 야기하지만 가독성의 향상을 통해 보다 나은 팀 프로젝트 활동을 수행할 것이다.

이때까지의 덧셈과 뺄셈에서와는 달리 곱셈에서는 많은 메모리의 할당과 실행시간이 오래 걸릴 것으로 예상되는데 이 문제점을 해결하기 위해서 최적화 측면에서 속도를 향상시킬 방안을 구상할 것 이다.

VII. 활동 일지 및 역할 분담

11 월 15 일 | 정기적 미팅 시간 및 장소 정하기

11 월 19 일 | 스택의 구조에 관련해 공부하고 파일에서 무한 수를 받는 코드를 DLL 실습을 이용하여 구현하는 알고리즘 구상

11 월 26 일 | DLL 로 입력받은 수를 후위표기법으로 바꾸는 스택의 알고리즘과 코딩

11 월 27 일 | 더하기 계산 알고리즘, 후위표기법 디버깅, 최적화 관련 학습 및 토의

11 월 29 일 | 두 수의 정수 덧셈 계산 구현 알고리즘, 다른 경우 상황 고려(정수 + 정수, 정수 + 소수, 소수 + 소수)

11 월 30 일 | 후위 표기법에서 괄호 구현 알고리즘 논의, 세 수 이상의 덧셈 구현 알고리즘 논의

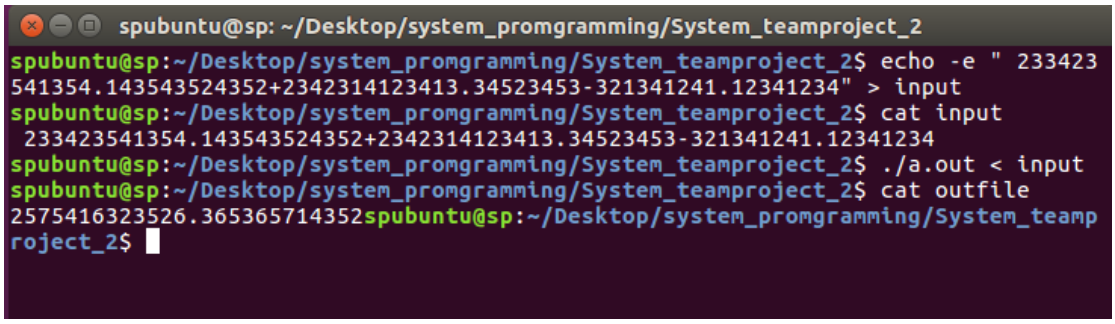
12 월 1 일 | 세 수 이상의 덧셈 및 뺄셈을 위한 delete_all 함수 구현

12 월 3 일 | 괄호 계산을 제외한 모든 덧셈, 뺄셈 계산 구현완료

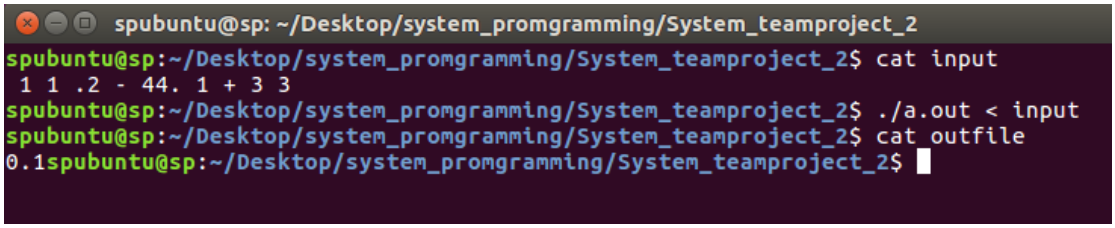
역할분담

권태현	Main programmer
김상준	Main programmer
김영민	Sub programmer
김영웅	Sub programmer & Report writer
김규진	Main Report writer

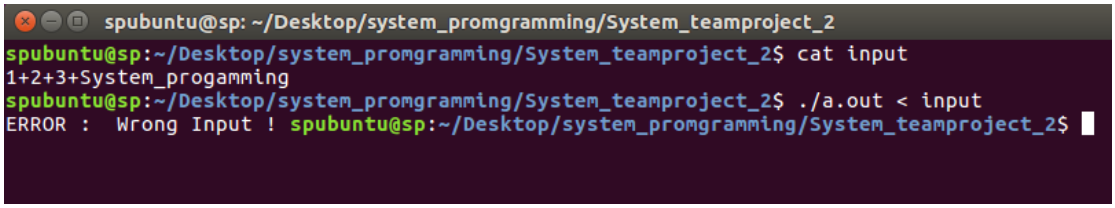
최종 결과물



1) 기본 덧셈, 뺄셈 계산



2) 자동 입력 처리



3) 예외처리