과제 요건 별 학습 사항

**[PBL1, 2]**

**1. 입력 문자열의 길이 제한은 없어야 한다.  
따라서, 동적 할당을 이용한 연결 리스트로 구현해야 한다.**

정적 할당은 프로그램 실행 전에만 할당이 가능하다. 실행 도중 지정한 값보다 입력 값이 크다면 오버플로우가 발생할 수 있고, 반대로 너무 큰 값을 할당하면 메모리 낭비가 크다. 무한 소수 계산기에서는 입력 크기에 제한이 없어야 하기 때문에 프로그램 실행 도중 메모리를 할당하는 동적 할당을 이용해야 한다. 이 때 숫자는 Double Linked List에 저장되는데, 한 글자씩 문자열로 입력을 받아 노드 하나에 저장하므로, 이 프로그램의 동적 할당에는 한 번에 같은 크기의 메모리를 여러 개 동적 할당하는 calloc 함수보다는 한 크기의 메모리 하나를 할당하는 malloc 함수가 적당하다.

**2. 파일로부터 입력 값을 받아온다.  
– 입력에 대한 예외처리를 해야 한다.**

**(유효하지 않은 입력 값에 대한 오류 처리)**

fopen 함수로 파일 포인터를 배정받아 직접 파일에서 읽어오는 것도 가능하다. 그러나 우리는 command line에서 redirection(<) 기호를 통해 입력을 받을 것이다. 이 경우 scanf()함수처럼 실행 후 사용자의 입력처럼 사용할 수 있다. 문자열에 길이 제한도 없어야 하고, 한 글자씩 읽어와 처리해야 하므로 getchar()함수를 이용할 것이다.

**입력에 대한 예외처리**

* 0~9 사이의 자연수, ‘(‘, ‘)’, ‘+’, ‘-‘, ‘\*’, ‘/’, ‘.’가 아닌 문자는 유효하지 않은 입력이다.
* 소수점으로 수가 시작하거나 끝날 수는 없다.
* 소수점 두 개가 연달아 올 수 없다.
* 여는 괄호와 닫는 괄호의 개수가 같아야 한다. (짝이 맞아야 한다.)
* 앞에서부터 하나씩 읽을 때 닫는 괄호의 개수가 여는 괄호의 개수보다 클 수 없다.
* ‘(-‘를 제외하고 연산자는 두 개 이상 연달아 올 수 없다. (입력은 중위 표기)
* ‘(-‘ 또는 ‘-‘로 시작하는 경우 음수로 저장한다.

위에 해당하는 경우, 예외 내용을 출력하고 return 0;을 통해 프로그램을 정지한다.

**3. 스택을 이용한 수식 표기법 변환 함수를 구현해야 한다.**

**- 중위 표기법(infix) -> 후위 표기법(postfix)**

0. 우선순위는 ‘+’ = ‘-‘ < ‘\*’ = ‘/’ 순이다. (괄호는 따로 처리한다.)

1. 입력 파일(중위 표기식)에서 한 글자씩 읽는다. (getchar())

2. 읽은 글자가 피연산자이면 바로 후위 표기식에 옮긴다.

3. 읽은 글자가 연산자이면

3-1. 왼쪽 괄호일 경우 스택에 넣는다.

3-2. 오른쪽 괄호일 경우 왼쪽 괄호를 꺼낼 때까지 스택에서 pop하여 후위 표기식으로 옮긴다. 왼쪽 괄호를 뽑으면 오른쪽 괄호와 함께 버리고, 스택에서 꺼내기를 끝낸다.

3-3. 나머지 연산자의 경우 그 연산자가 스택의 최상위 노드보다 우선순위가 작지 않을 때까지 스택의 최상위 노드를 pop하여 후위 표기식으로 옮긴다. 현재 연산자가 스택의 최상위 노드보다 우선순위가 높아졌다면, 현재 연산자를 스택에 Push한다.

4. 입력이 끝나면 스택에 남은 모든 연산자를 pop하여 후위 표기식에 옮겨 적는다.

**4. 덧셈 함수, 뺄셈 함수를 구현해야 한다.**

덧셈 함수와 뺄셈 함수 모두 두 수가 모두 양수일 때만 구현하면 된다.

나머지 경우는 덧셈과 뺄셈 함수를 오가며 결국 양수 두 개 간의 연산으로 바꿀 수 있다.

**- 덧셈 함수**

1. 소수점을 기준으로 양 쪽의 노드 개수를 같게 한다. 양 끝에 value가 0인 노드를 추가하는 방식이다.

2. tmp 노드를 2개 정의하여 두 수를 맨 뒤부터 접근한다. 소수점은 신경 쓰지 않는다.

3. 올림이 있는지 없는지 int형 변수를 통해 전 노드로 전달한다.

4. tmp 노드를 앞으로 옮겨가며, 두 node의 value와 뒤 노드에서 저장한 up을 더한 값을 value로 가지는 노드를 ans라는 double linked list의 맨 앞 노드로 추가한다. 이 값이 10이 넘을 경우는 Up 을 통해 다음 차례(앞 노드)로 전달하고, 일의 자리의 값만 저장한다.

5. 맨 앞 노드까지 진행하고, 반복문이 끝난 상태에서 up==1이면 value가 1인 노드를 ans의 맨 앞에 추가한다.

**- 뺄셈 함수**

0. 두 수 Num1과 Num2를 Num1>=Num2가 되게 한다. 두 수를 바꾼 경우는 ans->sign을 -1로 바꾼다.

1. 소수점을 기준으로 양 쪽의 노드 개수를 같게 한다. 양 끝에 value가 0인 노드를 추가하는 방식이다.

2. tmp 노드를 2개 정의하여 두 수를 맨 뒤부터 접근한다. 소수점은 신경 쓰지 않는다.

3. 버림이 있는지 없는지 int형 변수를 통해 전 노드로 전달한다.

4. tmp 노드를 앞으로 옮겨가며, 두 node의 value와 뒤 노드에서 저장한 down을 더한 값을 value로 가지는 노드를 ans라는 double linked list의 맨 앞 노드로 추가한다. 이 값이 음수일 경우 down 을 통해 다음 차례(앞 노드)로 전달하고, 앞에서 10을 가져와 더해 저장한다.

5. 맨 앞 노드까지 반복문을 통해 진행한다.

**5. 프로그램의 최적화란 무엇인가에 대한 학습을 진행한다.**

프로그램 최적화란 소프트웨어가 동일한 실행시 더 적은 메모리가 소요되는 것 또는 더 효과적으로 작동하는 것을 말한다. 최적화 방법으로는 비효율적인 알고리즘 개선, 반복되는 부분을 변환하기, 동적 변수 할당이 있다.

먼저 알고리즘 개선은 불필요한 if-else문 대신 switch-case문을 지향하는 것이다. If-else문을 사용하면 매 경우 처음부터 모든 경우를 찾아보게 되어 많은 메모리를 사용한다. 반면에 switch-case문을 사용하면 매 경우 한 번의 비교를 하여 찾아가기 때문에 훨씬 적은 메모리를 사용한다.

둘째로, 반복되는 부분이 있다면 코드를 중복 실행하지 않고 저장하여 다시 가져다 사용해 더욱 빠른 속도를 낼 수 있다. 예를 들면, 피보나치 수열에서 수많은 중복이 일어나는 함수를 그대로 실행한다면 1000번만 실행해도 엄청 오랜 시간이 걸린다. 중복되는 수를 저장하여 다시 사용한다면 메모리에서 손해를 보지만 훨씬 빠른 속도로 실행을 할 수 있다.

마지막으로 동적 변수 할당에는 C언어에서의 포인터 사용이 있다. 변수를 만들어 두는게 아니라, 사용할 때마다 할당해 준다면 메모리에 쓸모없이 저장되는 부분이 없어 메모리 사용이 줄어든다. 또한 오버플로우를 막을 수 있다는 장점도 있다.

프로그램 최적화에 대해 공부하며, 메모리 사용을 줄이면서 프로그램을 빠르게 하는 것이 쉬운 일이 아님을 알게 되었다. 프로그램 최적화를 통해 프로그램이 무조건 빨라지는 것이 아니라 오히려 느려질 수도 있고, 많은 부분이 코드를 새로 만들어야 할지도 모른다.

그리고 예전에는 보통의 메모리 용량이 지금처럼 크지 않아 최적화 중에서도 메모리를 줄이는 작업을 제일 중요하게 생각하였지만, 요즘에는 메모리가 예전보다는 저렴하고 기본 장착 메모리가 훨씬 커졌다. 이러한 점에서 최적화 작업 시 메모리와 시간 효율성에 대한 부분이 충돌한다면, 메모리 사용을 줄이기보다는 속도를 빠르게 하는 걸 더 중요시하게 되는 경향이다.

**[PBL3] 곱셈 함수를 구현해야 한다.**

연산하는 두 수를 Num1, Num2라고 할 때 Num1전체와 Num2 한 자리를 곱해서 Num2의 모든 자리를 곱한 값을 전부 더하는 식으로 구현한다. 예를 들면, 3.14\*8.9는 (3.14)\*8\*1+(3.14)\*9\*0.1과 같이 계산한다.

1. (numList \*ans의 초기값은 value==0인 numNode 하나로 이뤄진 numList이다.)
   1. 자릿수에 상관없이, numList 하나와 numNode 한 자리를 곱해 numList\*를 반환하는 함수를 작성한다. (tmp에 이 return값을 저장)
   2. numNode \*tmp를 Num2의 tail부터 앞쪽으로(prev) 옮기면서 반복문을 돌린다. 반복문 내에서 1의 함수를 각 tmp에 대해 호출한다.

return된 NumList를 덧셈 함수를 통해 ans에 더한다. ans = plus(ans,tmp);와 같이 구현한다. 이 때 덧셈과 같이 int형 변수 up을 통해 올림을 구현한다. (소수점은 신경 쓰지 않는다.)