

Infinity Calculator

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

| **2018044320 김규진**

| **2018044375 김상준**

| **2018044420 김영민**

| **2018044439 김영웅**

| **2018044302 권태현**

目 次

1. 개요
2. 입력 값 받기  
   ----1) 파일 입출력  
   ----2) 구현 설명
3. 문자열의 길이 제한  
   ----1) DLL(연결 리스트)  
   ----2) 구현 설명
4. 수식 표기법 변환

----1) 파일로부터 입력 값 받기(redireaction  
----2) 문자열의 길이 제한(DLL

1. 덧셈 함수 구현

----1) 파일로부터 입력 값 받기(redireaction  
----2) 문자열의 길이 제한(DLL

1. 뺄셈 함수 구현

----1) 파일로부터 입력 값 받기(redireaction  
----2) 문자열의 길이 제한(DLL

1. 프로그램 최적화 및 발전방향
2. 활동일지 및 역할분담

I. 개요

시스템 프로그래밍 수업에서 배운 파일에서 입력 값 받아오기, 스택, 연결리스트 등과 실습시간에 배운 실습과제 10주차까지를 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, 연산 식이Operator(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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | void getnumber(DLL \*list, FILE \*ifp) {  int count = 0;  char temp;  while (1) {  temp = getc(ifp);  if (temp == '\n') break;  if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-' || temp == '\*' || temp == '/') {  if (count == 0) {  if (temp == '+') {  파일에서 입력 값을 입력 받는 함수에서 ‘-3 + 5’와 같은 예외를 처리  temp = getc(ifp);  count++;  }  if (temp == '-') {  list->swh = 2;  temp = getc(ifp);  count++;  }  }  if (temp == '+' || temp == '-' || temp == '\*') list->i = list->i + 1;  append(list, newnode(temp));  count++;  ///////////////  temp = getc(ifp);  if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-' || temp == '\*' || temp == '/') {  printf("ERROR: Worng Input !");  24라인: 연산자의 연속  30라인: 알파벳이 입력  숫자 이외 값 입력을 처리 .  break;  }  else if (isblank(temp)) count++;  else if (isalpha(temp)) {  printf(" Wrong Input ! ");  break;  }  else if (isdigit(temp)) {  append(list, newnode(temp));  count++;  }  else {  printf(" Wrong Input ! ");  break;  }  //////////////  }  else if (isblank(temp)) count++;  else if (isalpha(temp)) {  printf(" Wrong Input ! ");  break;  }  else if (isdigit(temp)) {  append(list, newnode(temp));  count++;  }  else {  printf(" Wrong Input ! ");  break;  }  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info" \l "e" \t "_blank) |   *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

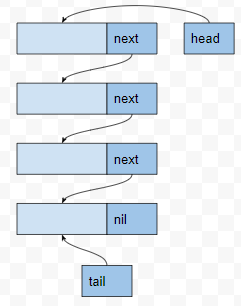
III. 문자열 길이 제한

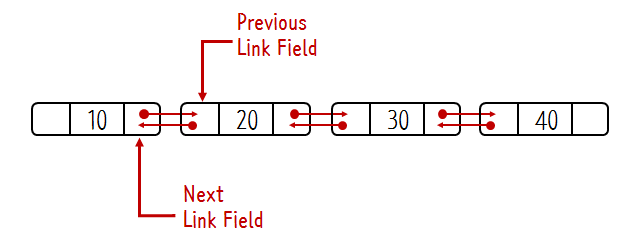
프로젝트에서 원하는 아주 큰 수를 처리하기 위해서 C언어에서 제공하는 기본 자료형에는 한계가 있다. 

위의 표를 참고하면, unsigned long long이 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 1234라는 수를 각 자리에 저장했다고 하면 1에서 다음에 있는 노드인 2값으로 이동하기 는next를 통해 쉽지만 반대로 2에서 1노드로는 이동하기가 어렵다는 점이다.

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

DLL은 양방향으로 연결되어 있기 때문에 다음 노드로 이동하는 것과 마찬가지로 이전 노드를 prev포인터를 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

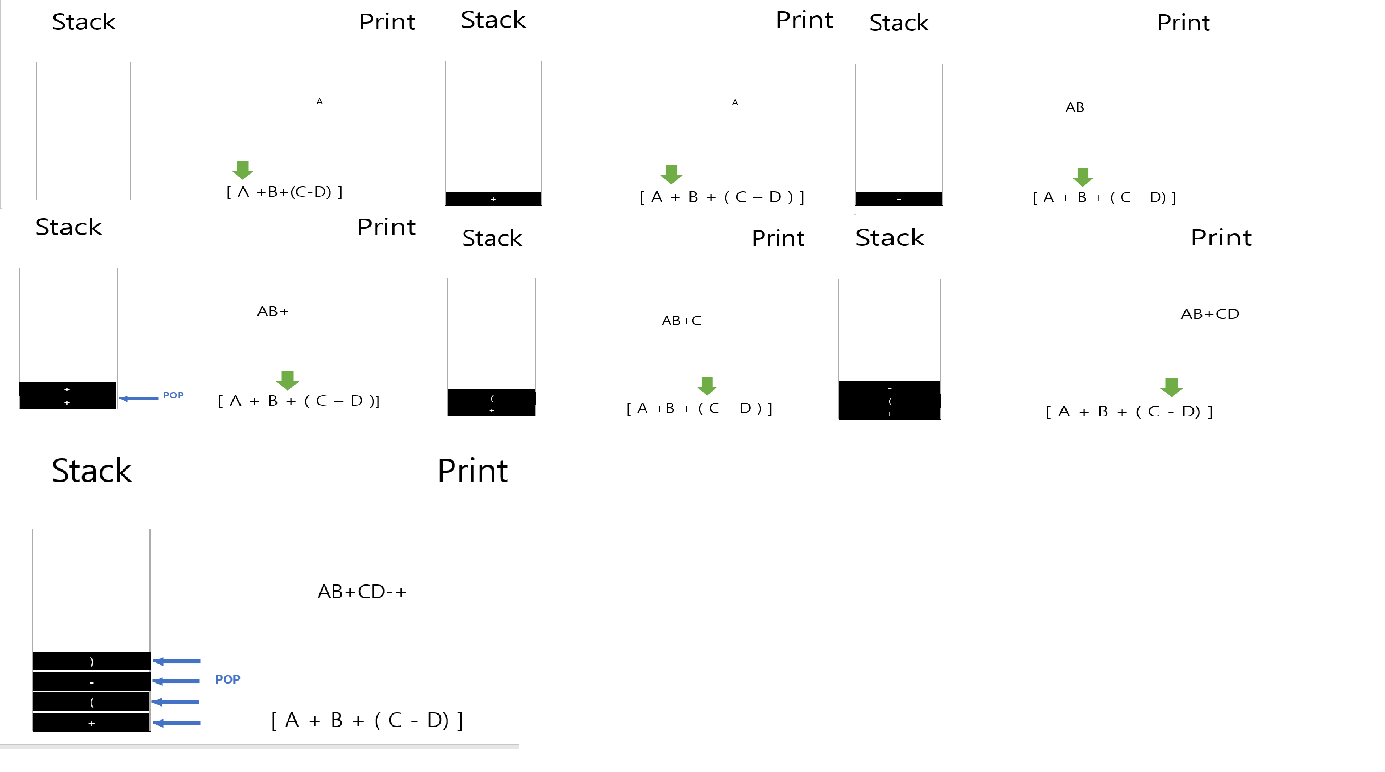
|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | typedef struct Node {  char val;  struct Node \*next;  struct Node \*prev;  } Node;  뒤에 쓰이는 변수들이지만  size\_1 은 소수점 자릿수를 저장하는 변수  size\_2은 자연수 자리수를 저장하는 변수  swh은 stack\_3의 부호표시를 위한 switch  i연산자의 개수의 변수이다.  typedef struct {  Node \*head;  int size;  int size\_1;  int size\_2;  int swh;  int i;  } DLL;  void append(DLL \*list, Node \*newnode) {  Node \*curr;  if (list->head == NULL) {  list->head = newnode;  }  else {  curr = list->head;  while (curr->next != NULL) {  curr = curr->next;  }  newnode->prev = curr;  curr->next = newnode;  }  list->size = list->size + 1;  }  void getnumber(DLL \*list) {  int count = 0;  while (1) {  char temp = getc (ifp);  if (temp == '\n') break;  else if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-' || temp == '\*' || temp == '/') {  if (count == 0) {  if (temp == '+') {  temp = getc (ifp);  count++;  }  if (temp == '-') {  list->swh = 2;  temp = getc (ifp);  count++;  }  }  if (temp == '+' || temp == '-' || temp == '\*') list->i = list->i + 1;  append(list, newnode(temp));  count++;  ///////////////  temp = getc (ifp);  if (temp == '.' || temp == '(' || temp == ')' || temp == '+' || temp == '-' || temp == '\*' || temp == '/') {  printf("ERROR: Worng Input !");  break;  }  else if (isblank(temp)) count++;  else if (isalpha(temp)) {  printf(" Wrong Input ! ");  break;  }  else if (isdigit(temp)) {  append(list, newnode(temp));  count++;  }  else {  printf(" Wrong Input ! ");  break;  }  }  else if (isblank(temp)) count++;  else if (isalpha(temp)) {  printf(" Wrong Input ! ");  break;  }  else if (isdigit(temp)) {  append(list, newnode(temp));  count++;  }  else {  printf(" Wrong Input ! ");  break;  }  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | int GreaterOpr(char opr1, char opr2)  {  if (opr1 == '\*') {  if (opr2 == '+' || opr2 == '-' || opr2 == ')')  return TRUE;  연산자의 우선순위를 결정하는 함수  opr1이 opr2보다 우선순위가 높으면 TRUE를 return  TRUE와 FALSE는 #define을 통해 1과 0으로 정의  else if (opr2 == '\*')  return TRUE;  else if (opr2 == '(')  return FALSE;  }  else if (opr1 == '+' || opr1 == '-') {  if (opr2 == '+' || opr2 == '-' || opr2 == ')')  return TRUE;  else if (opr2 == '\*' || opr2 == '(')  return FALSE;  }  else if (opr1 == '(') {  if (opr2 == ')')  return TRUE;  else  괄호를 포함한 연산자 우선순위에 따라 프로그래밍  return FALSE;  }else if (opr1 == ')')  return TRUE;  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

더 이상 append할 Node가 없을 때 스택의 남아있는 연산자를 모두 Pop해주는 함수

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | void POP\_all(DLL \*stack, DLL \*list\_1)  {  Node \*cur = stack->head;  while (cur->next != NULL) {  cur = cur->next;  }  while (1) {  if (cur->prev == NULL) {  append(list\_1, newnode(cur->val));  break;  }  else {  append(list\_1, newnode(cur->val));  cur = cur->prev;  }  }  free(stack);  stack = newDLL();  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

list\_1:후위표기법 변환 리스트

stack: 연산자 보관 리스트

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | void PushOrPop(DLL \*stack, char input\_opr, DLL \*list\_1)  { //Push 할건지 Pop할건지 판단하는 함수  while (1) {  int ssize = stack->size;  if (ssize == 0) {  append(stack, newnode(input\_opr));  ssize += 1;  GreatOpr함수를 이용  연산자를 어느 스택에 Push나 Pop을 할 것 인지 판단하는 함수  break;  }  else {  Node \*bigyo = stack->head;  while (bigyo->next != NULL)  bigyo = bigyo->next;  if (GreaterOpr(bigyo->val, input\_opr) == FALSE) {  append(stack, newnode(input\_opr));  ssize += 1;  break;  }  else {  if (ssize == 1) {  POP\_all(stack, list\_1);  append(stack, newnode(input\_opr));  ssize += 1;  break;  }  else if (ssize >= 2) {  append(list\_1, newnode(bigyo->val));  bigyo = bigyo->prev;  bigyo->next = NULL;  ssize -= 1;  }  }  }  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | void postfix(DLL \*list, DLL \*list\_1) {  DLL \*stack = newDLL();  Node \*curr = list->head;  while (1) {  if (curr->next == NULL) {  append(list\_1, newnode(curr->val));  최종 후위표기 변환함수  append(list\_1, newnode(' '));  break;  }  else if (isdigit(curr->val) || curr->val == '.')  {  append(list\_1, newnode(curr->val));  if (curr->next == NULL) break;  curr = curr->next;  }  else {  append(list\_1, newnode(' '));  PushOrPop(stack, curr->val, list\_1);  curr = curr->next;  }  }  POP\_all(stack, list\_1);  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |
|  |  |  |

후위표기를 보관하는 list\_1스택에 괄호를 지워주는 함수 // 8시42분 수정 요망

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | void delete\_par(DLL \*list\_1) {  Node \*curr = list\_1->head;  if (curr->val == ' ') deleteAt(list\_1, 0);  while (1) {  if (curr->next == NULL) {  if (curr->val == '(' || curr->val == ')') {  curr->prev->next = NULL;  }  break;  }  if (curr->val == '(' || curr->val == ')') {  curr->prev->next = curr->next;  curr->next->prev = curr->prev;  }  curr = curr->next;  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info" \l "e" \t "_blank) |

V 덧셈 & 뺄셈 보조 함수

후위 표기법 따라 연산하고 해당Node를 비우는 함수

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5 | void delete\_all(DLL \*list1) {  int a = list1->size;  for (int i = 0; i < a; i++)  deleteAt(list1, 0);  } | [cs](http://colorscripter.com/info#e) |



|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44. | int deleteAt(DLL \*list, int index) {  Node \*curr = list->head;  int count = 0;  if (list->size > 1) {  if (index == 0) {  list->head->next->prev = NULL;  list->head = list->head->next;  Index를 이용 빈 Node를 삭제하는 함수.  list->size = list->size - 1;  return 0;  }  while (count != index) {  count++;  curr = curr->next;  if (curr == NULL) {  printf("DELETE ERROR: Out of Bound.\n");  return 0;  }  }  if (count == index) {  if (curr->next == NULL) {  curr->prev->next = NULL;  list->size = list->size - 1;  return 0;  }  curr->prev->next = curr->next;  curr->next->prev = curr->prev;  list->size = list->size - 1;  }  }  else {  list->size = list->size - 1;  list->head->prev = NULL;  list->head->next = NULL;  list->head = NULL;  }  }  int insertAt(DLL \*stack\_3, int index, Node \*newnode) {  Node \*curr = stack\_3->head;  int count = 0;  if (index < 0) {  printf("INSERT ERROR: Out of Bound.\n");  return 0;  }  소수점 삽입을 위한 함수로 index에 소수점 삽입  if (index == 0) {  stack\_3->head = newnode;  curr->prev = newnode;  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](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  49  50  51 | void insert(DLL \*list\_1, DLL \*list\_3) {  append(list\_3, newnode(' '));  Node \*curr = list\_1->head;  Node \*curr\_1 = list\_3->head;  while (1) {  if (curr->val == '+' || curr->val == '-' || curr->val == '\*') {  curr = curr->next;  계산된 DLL을 후위표기법 연산 식에 병합  break;  }  else {  curr = curr->next;  }  }  while (1) {  if (curr\_1->next == NULL) break;  else curr\_1 = curr\_1->next;  }  while (1) {  if (curr->next == NULL) {  append(list\_3, newnode(curr->val));  break;  }  append(list\_3, newnode(curr->val));  curr = curr->next;  }  }  int insertAt\_int(DLL \*list, int index, Node \*newnode) {  int count = 0;  Node \*curr = list->head;  if (index == 0) {  list->head = newnode;  curr->prev = newnode;  list->head->next = curr;  return 0;  }  while (count != index) {  count++;  curr = curr->next;  if (curr->next == NULL) break;  }  if (curr->next == NULL) {  newnode->prev = curr;  curr->next = newnode;  return 0;  }  newnode->prev = curr->prev;  newnode->next = curr;  curr->prev->next = newnode;  curr->prev = newnode;  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | void plus\_zero(DLL \*list\_3) {  Node \*curr = list\_3->head;  while (1) {  if (curr->val == '.') {  if (curr->prev == NULL) {  insertAt(list\_3, 0, newnode('0'));  break;  }  if (curr->prev->val == '-') {  insertAt(list\_3, 1, newnode('0'));  }  예를 들어 ‘1 – 1.1’과 같은 연산 식에서 답이 ‘–.1’이 도출됨.  해결을 위해 0을 추가해주는 코드  }  if (curr->next == NULL) break;  curr = curr->next;  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | void plus\_change(DLL \*list\_1) { // 앞의수가 -일경우 뒤에있는 +를 -로 바꿔주기 위한 함수  Node \*curr = list\_1->head;  앞의 수가 음수 일 경우에 뒤에 있는 연산자 ‘-’를 ‘+‘로 바꾸어 주는 함수  if (curr->val == '-') {  while (1) {  if (curr->next == NULL) break;  curr = curr->next;  if (curr->val == '+') {  curr->val = '-';  list\_1->swh = 2;  deleteAt(list\_1, 0);  break;  }  else if (curr->val == '-') {  curr->val = '+';  deleteAt(list\_1, 0);  list\_1->swh = 3;  break;  }  }  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

DLL노드의 앞뒤를 바꾸어 주는 함수

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | void reverse(DLL \*list, DLL \*list\_1) { //  Node \*curr = list->head;  while (1) {  if (curr->next == NULL) break;  else curr = curr->next;  }  while (1) {  if (curr->prev == NULL) {  append(list\_1, newnode(curr->val));  break;  }  else {  append(list\_1, newnode(curr->val));  curr = curr->prev;  }  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | void size\_check(DLL \*list\_1) {  Node \*curr = list\_1->head;  list\_1->size = 0;  while (1) {if (curr->next == NULL) {  list\_1->size = list\_1->size + 1;  break;  }  else {  list\_1->size = list\_1->size + 1;  curr = curr->next;  }  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e)  List\_1 의size를 체크해주는 함수 |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122 | void zero(DLL \*stack\_1, DLL \*stack\_2) {  Node \*curr = stack\_1->head;  Node \*curr\_1 = stack\_2->head;  피연산자들의 정수와 소수의 자릿수를 0을 넣어 맞춰주는 함수  DLL \*stack\_3 = newDLL();  int a = 0;  int b = 0;  int c = 0;  int d = 0;  while (1) {  if (curr->val == '.') break;  else if (curr->next == NULL) {  c++;  break;  }  else {  c++;  curr = curr->next;  }  }  while (1) {  if (curr->next == NULL) break;  else {  a++;  curr = curr->next;  }  }  while (1) {  if (curr\_1->val == '.') break;  else if (curr\_1->next == NULL) {  d++;  break;  }  else {  d++;  curr\_1 = curr\_1->next;  }  }  while (1) {  if (curr\_1->next == NULL) break;  else {  b++;  curr\_1 = curr\_1->next;  }  }  stack\_1->size\_2 = c;  stack\_2->size\_2 = d;  if (a == 0 && b != 0) {  append(stack\_1, newnode('.'));  }  if (b == 0 && a != 0) append(stack\_2, newnode('.'));  if (c > d) {  ‘c’ 와 ’d’ 비교를 통해 정수부분 자릿수를 맞춤  int f = c - d;  for (int i = 0; i < f; i++) {  insertAt\_int(stack\_2, 0, newnode('0'));  }  }  else if (d > c) {  int f = d - c;  for (int i = 0; i < f; i++) {  insertAt\_int(stack\_1, 0, newnode('0'));  }  }  else {  }  if (a > b) {  int c = a - b;  for (int i = 0; i < c; i++) {  append(stack\_2, newnode('0'));  }  stack\_1->size\_1 = a;  stack\_2->size\_1 = a;  }  else if (b > a) {  int c = b - a;  for (int i = 0; i < c; i++) {  append(stack\_1, newnode('0'));  }  stack\_1->size\_1 = b;  stack\_2->size\_1 = b;  }  else {  stack\_1->size\_1 = a;  stack\_1->size\_1 = b;  }  if (c < d) {  copy\_1(stack\_3, stack\_1);  delete\_all(stack\_1);  copy\_1(stack\_1, stack\_2);  delete\_all(stack\_2);  copy\_1(stack\_2, stack\_3);  delete\_all(stack\_3);  stack\_1->swh = 1;  stack\_2->swh = 1;  }  else if (c == d) {  Node \*curr\_2 = stack\_1->head;  Node \*curr\_3 = stack\_2->head;  while (1) {  if (curr\_2->val == '.' && curr\_3->val == '.') {  curr\_2 = curr\_2->next;  curr\_3 = curr\_3->next;  }  int m = curr\_2->val - 48;  int n = curr\_3->val - 48;  if (n > m) {  copy\_1(stack\_3, stack\_1);  delete\_all(stack\_1);  copy\_1(stack\_1, stack\_2);  delete\_all(stack\_2);  copy\_1(stack\_2, stack\_3);  delete\_all(stack\_3);  stack\_1->swh = 1;  stack\_2->swh = 1;  break;  }  if (curr\_2->next == NULL && curr\_3->next == NULL) break;  curr\_2 = curr\_2->next;  curr\_3 = curr\_3->next;  }  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info#e) |

V. 덧셈 뺄셈 함수 구현

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211 | void cal(DLL \*list, DLL \*stack\_3) {  List: 후위표기법으로 바뀐 식  stack\_3: 연산의 값을 저장하는 리스트.  DLL \*stack\_1 = newDLL(); // 숫자1  DLL \*stack\_2 = newDLL(); // 숫자2  Node \*curr = list->head;  while (1) { // + - 만날 때 까지 노드 움직임  if (curr->val == '+' || curr->val == '-') {  curr = curr->prev;  curr = curr->prev;  break;  }  else {  curr = curr->next;  }  }  while (1) { //첫 번째 인자 수 맨 앞 빈칸  if (curr->prev == NULL || curr->val == ' ') {  break;  연산자를 만날 때까지 반복  피 연산자들을 ‘ ‘로 구분 후 저장  }  else {  curr = curr->prev;  }  }  curr = curr->next;  while (1) { //첫 번째 수 저장  if (curr->val == ' ') {  curr = curr->prev;  break;  }  else {  append(stack\_2, newnode(curr->val));  curr = curr->next;  }  }  while (1) { //첫 번째 수 넘어감  if (curr->val == ' ' || curr->prev == NULL) {  curr = curr->prev;  break;  }  else {  curr = curr->prev;  }  }  while (1) { //두 번째수 앞 빈칸까지  if (curr->val == ' ' || curr->prev == NULL) {  break;  }  else {  curr = curr->prev;  }  }  while (1) { //두번째 수 저장  if (curr->val == ' ') {  break;  }  else {  append(stack\_1, newnode(curr->val));  curr = curr->next;  }  }  zero(stack\_1, stack\_2);  if (list->swh == 2) {  stack\_1->swh = 2;  stack\_2->swh = 2;  list->swh = 0;  }  if (list->swh == 1) {  stack\_1->swh = 1;  stack\_2->swh = 1;  list->swh = 0;  }  if (list->swh == 3) {  stack\_1->swh = 3;  stack\_2->swh = 3;  list->swh = 0;  }  while (1) {  if (curr->val == '+' || curr->val == '-' || curr->val == '\*') {  break;  }  else {  curr = curr->next;  }  }  if (curr->val == '+') {  Node \*curr\_1 = stack\_1->head;  Node \*curr\_2 = stack\_2->head;  while (1) { // 노드 끝까지 보냄  if (curr\_1->next == NULL) break;  else curr\_1 = curr\_1->next;  }  while (1) { // 노드 끝까지 보냄  if (curr\_2->next == NULL) break;  else curr\_2 = curr\_2->next;  count : 올림수  ‘a’ & ’b’ : char val 를 int val로 변환  }  int count = 0;  int a = curr\_1->val - 48;  int b = curr\_2->val - 48;  int c;  while (1) { // + 계산시작  // 수가 끝낫을 경우 0으로 처리  if (curr\_1->val == '.') {  curr\_1 = curr\_1->prev;  }  if (curr\_2->val == '.') {  curr\_2 = curr\_2->prev;  }  a = curr\_1->val - 48;  b = curr\_2->val - 48;  c = a + b;  if (count == 1) {  c++;  count = 0;  if (c >= 10) {  c = c - 10;  count = 1;  }  }  else {  if (c >= 10) {  c = c - 10;  count = 1;  }  }  if (curr\_1 != NULL) curr\_1 = curr\_1->prev;  if (curr\_2 != NULL) curr\_2 = curr\_2->prev;  c = c + 48;  append(stack\_3, newnode(c));  if (curr\_1 == NULL && curr\_2 == NULL) {  if (count == 1) {  append(stack\_3, newnode('1'));  count = 0;  자리수가 올라간 경우  }  break;  }  } // while문  int d = stack\_1->size\_1;  if (d > 0) insertAt(stack\_3, d, newnode('.'));  if (stack\_1->swh == 1) stack\_3->swh = stack\_1->swh;  if (stack\_1->swh == 3) append(stack\_3, newnode('-'));// -3 5 - 인경우  stack\_1->swh = 0;  stack\_2->swh = 0;  stack\_3->swh = 0;  } // if +문에 걸림  //// -계산  else if (curr->val == '-') {  Node \*curr\_1 = stack\_1->head;  Node \*curr\_2 = stack\_2->head;  while (1) { // 노드 끝까지 보냄  if (curr\_1->next == NULL) break;  else curr\_1 = curr\_1->next;  }  while (1) { // 노드 끝까지 보냄  if (curr\_2->next == NULL) break;  else curr\_2 = curr\_2->next;  }  int count = 0;  int a = curr\_1->val - 48;  int b = curr\_2->val - 48;  int c;  while (1) {  if (curr\_1->val == '.') {  curr\_1 = curr\_1->prev;  }  if (curr\_2->val == '.') {  curr\_2 = curr\_2->prev;  }  a = curr\_1->val - 48;  b = curr\_2->val - 48;  c = a - b;  if (count == 1) {  c--;  count = 0;  }  if (c < 0) {  c = c + 10;  count = 1;  }  if (curr\_1 != NULL) curr\_1 = curr\_1->prev;  if (curr\_2 != NULL) curr\_2 = curr\_2->prev;  c = c + 48;  append(stack\_3, newnode(c));  if (curr\_1 == NULL && curr\_2 == NULL) {  break;  }  }  int d = stack\_1->size\_1;  stack\_3->swh = stack\_1->swh;  if (d > 0) insertAt(stack\_3, d, newnode('.'));  자연수의 앞자리가 0일 경우에 0삭제  Node \*curr\_3 = stack\_3->head;  while (1) {  if (curr\_3->next == NULL) break;  else curr\_3 = curr\_3->next;  }  while (1) {  if (curr\_3->val != '0') break;  else {  curr\_3 = curr\_3->prev;  curr\_3->next = NULL;  }  }  ////// stack\_3의 값이 -일경우  if (stack\_3->swh == 1) append(stack\_3, newnode('-'));  stack\_1->swh = 0;  stack\_2->swh = 0;  stack\_3->swh = 0;  }  }  *[Colored by Color Scripter](http://colorscripter.com/info" \l "e" \t "_blank)* | [cs](http://colorscripter.com/info" \l "e" \t "_blank) | |

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 volatile이다. 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 volatile로 선언해야 한다. 그렇게 volatile로 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 것 이다.

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

11월 15일 | 정기적 미팅 시간 및 장소 정하기

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

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

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

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

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

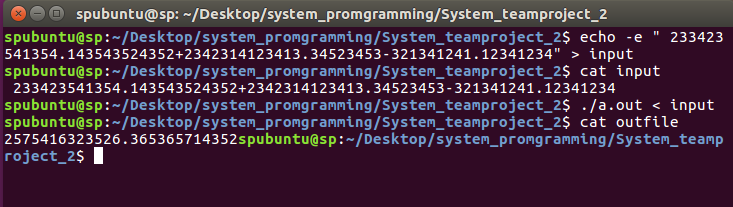
12월 1일 | 세 수 이상의 덧셈 및 뺼셈을 위한 delete\_all 함수 구현

12월 3일 | 괄호 계산을 제외한 모든 덧셈, 뺄셈 계산 구현완료

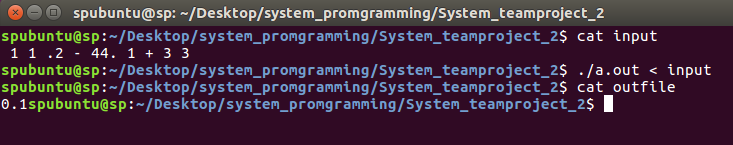
**역할분담**

|  |  |
| --- | --- |
| **권태현** | **Main programmer** |
| **김상준** | **Main programmer** |
| **김영민** | **Sub programmer** |
| **김영웅** | **Sub programmer & Report writer** |
| **김규진** | **Main Report writer** |

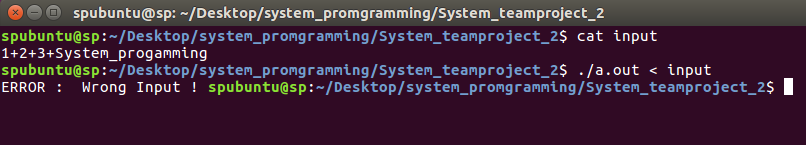
**최종 결과물**

****

1. **기본 덧셈, 뺄셈 계산**

****

1. **자동 입력 처리**

****

1. **예외처리**