

# Data Structure

실습 2

# 0. 이번 주 실습 내용

---

- **Linked List**
  - List
  - Array List
  - Linked List
  - Linked List 실습
  - Univariate Polynomial Multiplication(과제)

# 0. 재귀함수 Review

- 피보나치 수열 구하기
- $F(n) = F(n-1) + F(n-2)$  for  $n > 1$
- $F(0) = 0$
- $F(1) = 1$

```
1  #include <stdio.h>
2
3  int f(int n){
4      [REDACTED]
5      [REDACTED]
6      [REDACTED]
7  }
8
9  int main(){
10     int N = 10;
11     printf("%d", f(N));
12 }
```

# 0. 재귀함수 Review

- 피보나치 수열 구하기
- $F(n) = F(n-1) + F(n-2)$  for  $n > 1$
- $F(0) = 0$
- $F(1) = 1$

```
1  #include <stdio.h>
2
3  int f(int n){
4      if(n == 0) return 0;
5      else if(n == 1) return 1;
6      else return f(n-1) + f(n-2);
7  }
8
9  int main(){
10     int N = 10;
11     printf("%d", f(N));
12 }
```

- 입력  $n$ 에 대한 예제 코드의 시간복잡도?  
 $O(2^n)$

# 0. 재귀함수 Review

- 피보나치 수열 구하기
- $F(n) = F(n-1) + F(n-2)$  for  $n > 1$
- $F(0) = 0$
- $F(1) = 1$
- 입력  $n$ 에 대한 예제 코드의 시간복잡도?  
 $O(n)$

```
5 int f(int n, int* fs){
6     if(n == 0) return 0;
7     else if(n == 1) return 1;
8     else if(fs[n] > 0) return fs[n];
9
10    fs[n] = f(n-1, fs) + f(n-2, fs);
11    return fs[n];
12 }
13
14 int main(){
15     int N = 10;
16     int i;
17
18     int *fs = (int*)malloc((N+1)*sizeof(int));
19     for(i = 0; i < N; i++) fs[i] = -1;
20
21     printf("%d", f(N, fs));
22
23     free(fs);
24
25     return 0;
26 }
```

# 1. List

- 정의: 자료를 순서(한 줄)대로 저장하는 자료구조
- 구조가 단순하기 때문에 널리 사용되는 자료구조
- 여러 개의 자료가 일직선으로 연결된 '선형 구조'



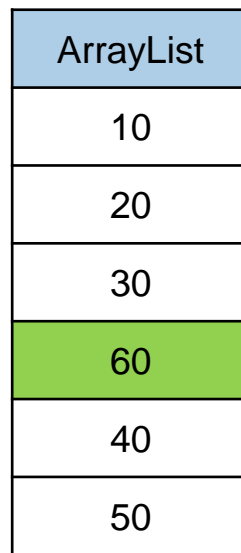
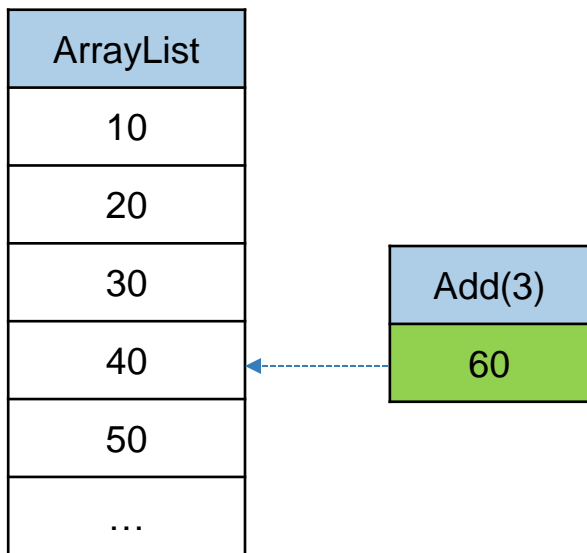
## 2. Array List

- 배열을 구성하는 원소들이 순서대로 연속되어 저장
- 논리적 순서와 물리적 순서(실제 메모리에 저장된 순서)가 같다
- Data에 접근하기가 쉽다
  - Ex) list[3] (Time Complexity:  **$O(1)$** )

0x1010	List[0]	10
0x1011	List[1]	20
0x1012	List[2]	30
0x1013	List[3]	40
0x1014	List[4]	50

## 2. Array List

- Data 삽입 Example (Add or Insert)

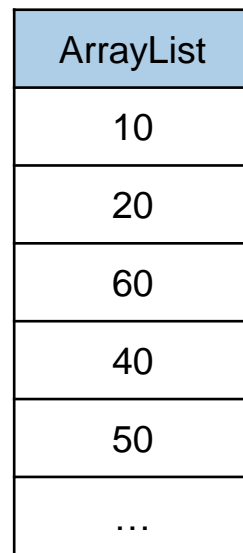
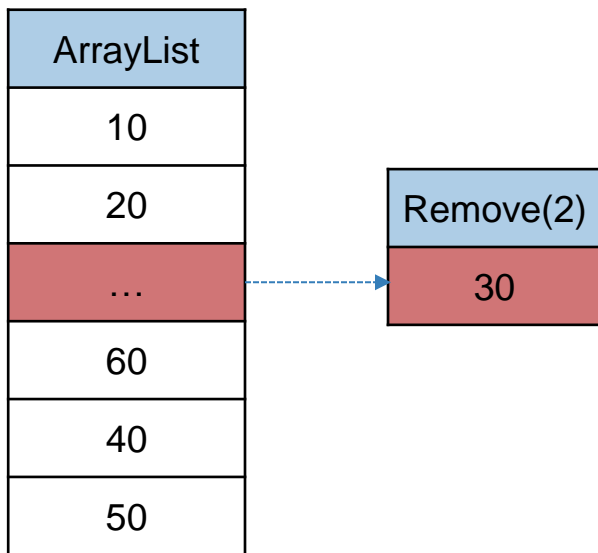


Time Complexity  
 $O(N)$



## 2. Array List

- Data 삭제 Example (Remove or Delete)

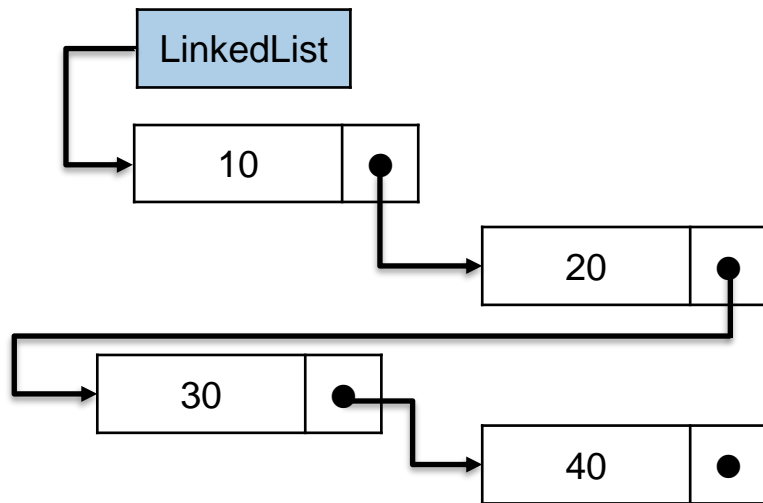


Time Complexity  
 **$O(N)$**

# 3. Linked List

- 포인터를 이용하여 물리적으로 멀리 떨어져 있는 자료들을 순서대로 연결함
- 논리적인 순서는 순차적이지만 물리적 순서(메모리에 저장된 위치)는 순서대로 인접해 있지 않다.

	ArrayList
0	10
1	20
2	30
3	40
4	50



# 3. Linked List

## • 구성 요소

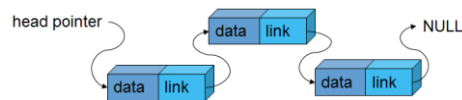
- **Node** : Data & Link로 구성된 기본 단위
  - Data: 실제 데이터를 저장하는 영역
  - Link: 그 다음 Node의 위치를 저장하는 영역



## • 종류

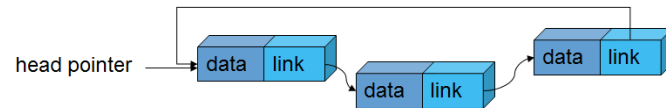
### • Single Linked List

- Node가 Data와 1개의 Link로 구성
- Link는 다음 Node의 위치를 저장



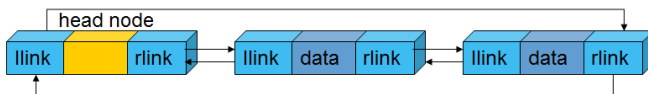
### • Circular Linked List

- Linked List가 원형 구조를 이룸
- 마지막 Node의 Link가 처음 Node 위치를 저장



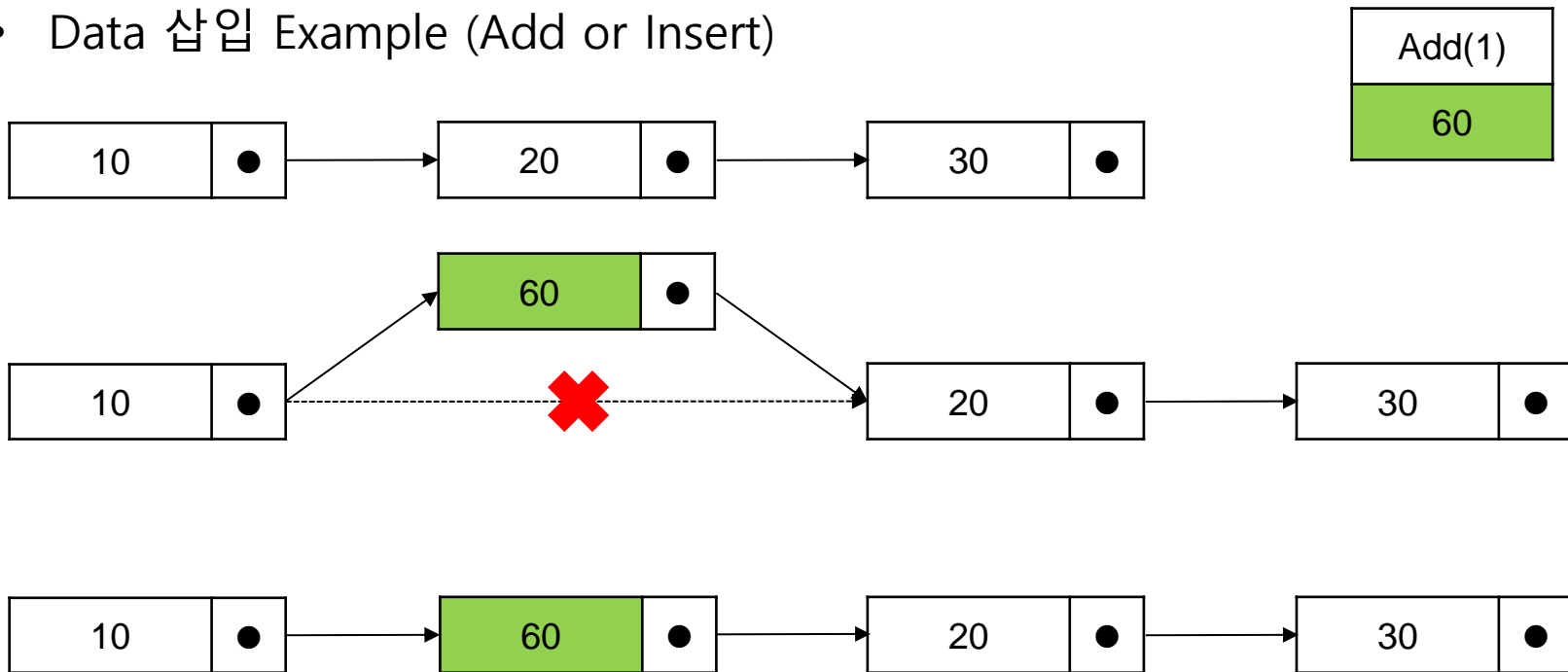
### • Doubly Linked List

- Node에 Data와 2개의 Link로 구성
- 2개의 Link는 각각 다음/이전 Node의 위치를 저장



# 3. Linked List

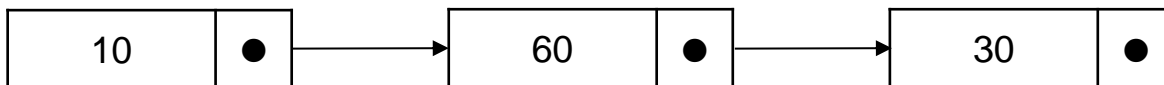
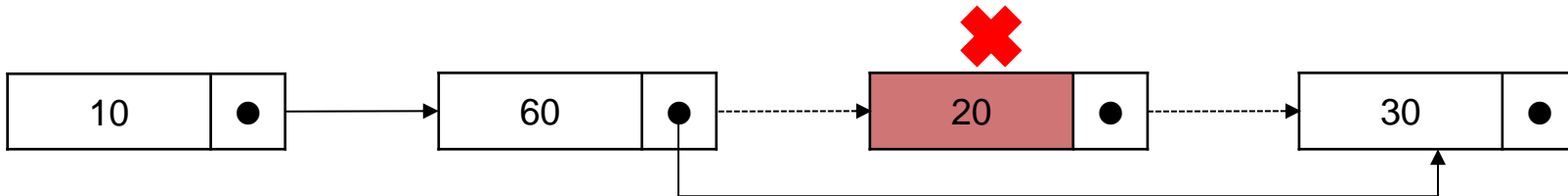
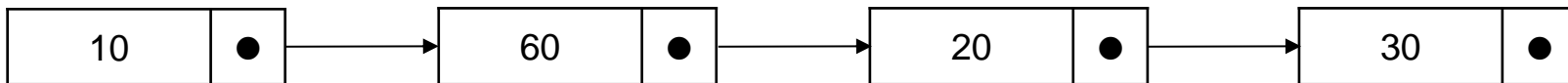
- Data 삽입 Example (Add or Insert)



# 3. Linked List

- Data 삭제 Example (Remove or Delete)

Remove(2)

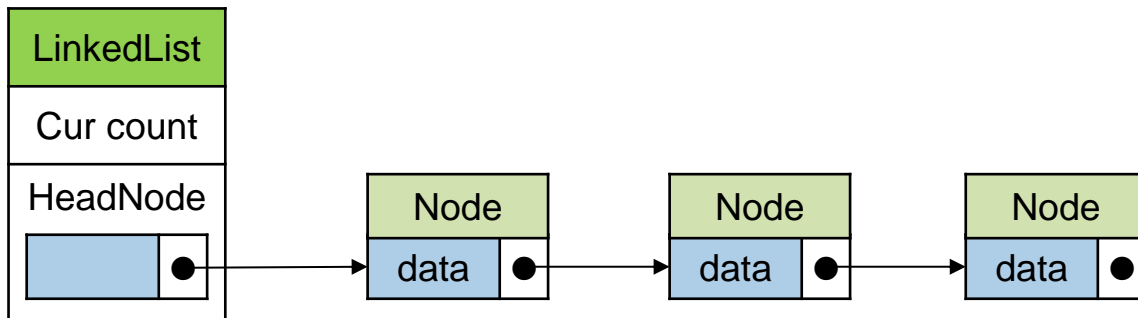


# 3. Linked List

---

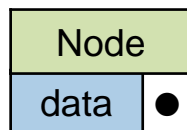
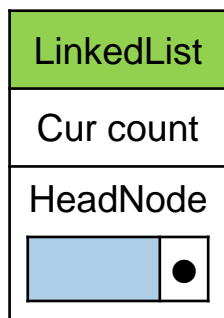
장점	단점
<ul style="list-style-type: none"><li>• 동적인 크기 조작이 가능</li><li>• 데이터 추가/삭제 연산 용이</li><li>• 데이터 추가/삭제 시 추가적인 데이터 이동 연산 불필요</li><li>• 여러가지 자료구조로 변형하기 쉬움 (Stack, Queue...)</li></ul>	<ul style="list-style-type: none"><li>• 데이터 탐색 연산이 어려움: <math>O(N)</math></li><li>• 구현이 어려움</li></ul>

### 3. Linked List - 실습



# 3. Linked List - 실습

- List ADT(Abstract Data Type)정의



```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define FALSE -1
6  #define TRUE 1
7
8  //List 의 기본 구성단위인 Node
9  typedef struct Node {
10     int data;
11     struct Node* nextNode;
12 }Node;
13
14 typedef struct LinkedList {
15     int curCount;    //현재 List에 들어있는 Node의 갯수
16     Node headNode;  //List의 시작 Node
17 }LinkedList;
18
19 int addNode(LinkedList* pList, int pos, int data);
20 int removeNode(LinkedList* pList, int pos);
21 void showNode(LinkedList* pList);
22 int isEmpty(LinkedList* pList);
23 int findPos(LinkedList* pList, int data);
24 void makeEmpty(LinkedList* pList);
    
```



### 3. Linked List - 실습

- Main 함수 부분(실행 Process)
  - Singular Linked List 생성
  - 여러가지 값을 갖는 Node들을 추가
  - 현재 Linked List의 상태를 출력
  - Linked List에서 특정 위치의 Node 삭제
  - Linked List에서 data값을 가진 Node 검색
  - Linked List 초기화

```
26 int main()  
27 {  
28     int pos;  
29     LinkedList* linkedList = (LinkedList*)malloc(sizeof(LinkedList));  
30     linkedList->curCount = 0;  
31     linkedList->headNode.nextNode = NULL;  
32  
33  
34     showNode(linkedList);  
35     addNode(linkedList, 0, 10);  
36     addNode(linkedList, 5, 100);  
37     addNode(linkedList, 1, 20);  
38     addNode(linkedList, 2, 30);  
39     addNode(linkedList, 1, 50);  
40  
41     showNode(linkedList);  
42  
43     removeNode(linkedList, 1);  
44     showNode(linkedList);  
45  
46     pos = findPos(linkedList, 30);  
47     printf("the location of node with data '30': %d\n", pos);  
48  
49     makeEmpty(linkedList);  
50     showNode(linkedList);  
51     return 0;  
52 }  
53
```

# 3. Linked List - 실습

- Linked List 출력 함수 구현

```

125 void showNode(LinkedList* pList)
126 {
127     int i = 0;
128     Node *pNode = NULL;
129
130     if(pList == NULL)
131     {
132         printf("showNode() error\n");
133         return ;
134     }
135
136     printf("현재 Node 개수 : %d \n", pList->curCount);
137     pNode = pList->headNode.nextNode;
138     //pNode가 Linked List의 마지막 노드까지 이동하면서 출력
139     while ( )
140     {
141         printf("[%d]\n", pNode->data);
142         //
143     }
144     printf("-----\n");
145 }

```

- Linked List 내부 Node가 존재하는 지 확인하는 함수 구현

```

147 int isEmpty(LinkedList* pList)
148 {
149     if (pList == NULL)
150     {
151         printf("isEmpty() error\n");
152         return -1;
153     }
154     //head 노드가 가리키는 next 노드가 존재하는가
155     if ( )
156         return TRUE;
157     else
158         return FALSE;
159 }

```

# 3. Linked List - 실습

- Linked List 출력 함수 구현

```

125 void showNode(LinkedList* pList)
126 {
127     int i = 0;
128     Node *pNode = NULL;
129
130     if(pList == NULL)
131     {
132         printf("showNode() error\n");
133         return ;
134     }
135
136     printf("현재 Node 개수 : %d \n", pList->curCount);
137     pNode = pList->headNode.nextNode;
138     //pNode가 Linked List의 마지막 노드까지 이동하면서 출력
139     while (pNode != NULL)
140     {
141         printf("[%d]\n", pNode->data);
142         pNode = pNode->nextNode;
143     }
144     printf("-----\n");
145 }

```

- Linked List 내부 Node가 존재하는 지 확인하는 함수 구현

```

147 int isEmpty(LinkedList* pList)
148 {
149     if (pList == NULL)
150     {
151         printf("isEmpty() error\n");
152         return -1;
153     }
154     //head 노드가 가리키는 next 노드가 존재하는가
155     if (pList->headNode.nextNode == NULL)
156         return TRUE;
157     else
158         return FALSE;
159 }

```

# 3. Linked List - 실습

- data값을 가지는 node탐색 함수 구현

```

162 int findPos(LinkedList* pList, int data)
163 {
164     int pos = 0;
165     Node *pNode = NULL;
166
167     if (pList == NULL)
168     {
169         printf("findPos() error\n");
170         return FALSE;
171     }
172
173     pNode = pList->headNode.nextNode;
174     //마지막 노드까지 탐색
175     while ( )
176     {
177         //노드의 data가 일치한다면 해당 위치 반환
178         [redacted]
179
180         //찾지 못할 경우 위치를 증가시키면서 pNode를 이동
181         [redacted]
182     }
183
184     return FALSE;
185 }

```

- Linked List 초기화 함수 구현

```

187 void makeEmpty(LinkedList* pList)
188 {
189     Node* pDummyNode = NULL, *pTmpNode=NULL;
190     if (pList != NULL)
191     {
192         pTmpNode = pList->headNode.nextNode;
193         //연결되어 있는 모든 노드들을 탐색
194         while ( [redacted] )
195         {
196             //Dummy 노드는 지우기 위한 노드
197             //Tmp 노드는 이동하기 위한 노드
198             [redacted]
199
200             [redacted]
201         }
202         [redacted]
203     }
204 }
205
206

```

# 3. Linked List - 실습

- data값을 가지는 node탐색 함수 구현

```

162 int findPos(LinkedList* pList, int data)
163 {
164     int pos = 0;
165     Node *pNode = NULL;
166
167     if (pList == NULL)
168     {
169         printf("findPos() error\n");
170         return FALSE;
171     }
172
173     pNode = pList->headNode.nextNode;
174     //마지막 노드까지 탐색
175     while (pNode != NULL)
176     {
177         //노드의 data가 일치한다면 해당 위치 반환
178         if (pNode->data == data)
179             return pos;
180         //찾지 못할 경우 위치를 증가시키면서 pNode를 이동
181         pos++;
182         pNode = pNode->nextNode;
183     }
184     return FALSE;
185 }

```

- Linked List 초기화 함수 구현

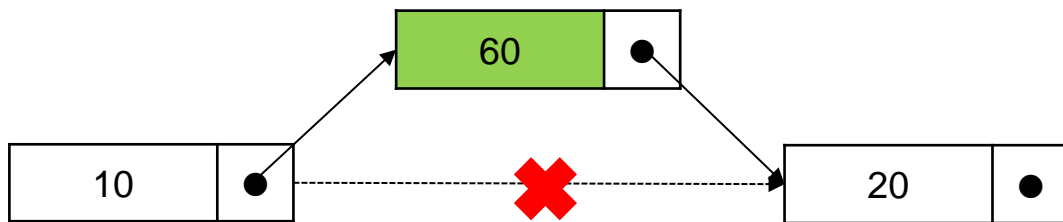
```

187 void makeEmpty(LinkedList* pList)
188 {
189     Node* pDummyNode = NULL, *pTmpNode=NULL;
190     if (pList != NULL)
191     {
192         pTmpNode = pList->headNode.nextNode;
193         //연결되어 있는 모든 노드들을 탐색
194         while (pTmpNode != NULL)
195         {
196             //Dummy 노드는 지우기 위한 노드
197             //Tmp 노드는 이동하기 위한 노드
198             pDummyNode = pTmpNode;
199             pTmpNode = pTmpNode->nextNode;
200             free(pDummyNode);
201         }
202         pList->headNode.nextNode = NULL;
203     }
204 }
205
206

```

### 3. Linked List - 실습

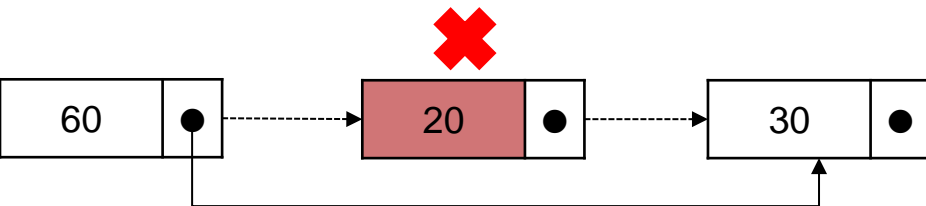
- Node 추가 함수 구현



```
54 int addNode(LinkedList* pList, int pos, int data)
55 {
56     int i = 0;
57     Node* pNewNode = NULL, *pTmpNode = NULL;
58     if (pList == NULL)
59     {
60         printf("addNode() error1 \n");
61         return FALSE;
62     }
63     if (pos < 0 || pos > pList->curCount)
64     {
65         printf("addNode() error2: 추가 범위 초과 \n");
66         return FALSE;
67     }
68
69     pNewNode = (Node*)malloc(sizeof(Node));
70     if (!pNewNode)
71     {
72         printf("addNode() error3 \n");
73         return FALSE;
74     }
75
76     pNewNode->data = data;
77     pNewNode->nextNode = NULL;
78
79     //추가될 위치 직전 노드로 이동
80
81
82
83
84     //추가 노드의 nextNode = 직전 노드의 nextNode
85     //직전 노드의 nextNode = 추가 노드의 주소
86
87
88
89     return TRUE;
90 }
91
```

# 3. Linked List - 실습

- Node 삭제 함수 구현



```

92 int removeNode(LinkedList* pList, int pos)
93 {
94     int i = 0;
95     Node* pDelNode = NULL, *pTmpNode = NULL;
96
97     if (pList == NULL)
98     {
99         printf("removeNode() error1\n");
100        return FALSE;
101    }
102
103    if (pos < 0 || pos > pList->curCount)
104    {
105        printf("removeNode() error2: 삭제 범위 초과\n");
106        return FALSE;
107    }
108
109    //삭제될 노드 직전 위치로 이동
110    pTmpNode = &(pList->headNode);
111    for (i = 0; i < pos; i++)
112        pTmpNode = pTmpNode->nextNode;
113
114    //삭제할 노드 = 직전 노드의 nextNode
115    //직전 노드의 nextNode = 삭제할 노드의 nextNode
116
117
118
119
120
121
122
123 }

```

### 3. Linked List - 실습

- 실행 결과

```
C:\ C:\WINDOWS\system32\cmd.exe  
현재 Node 개수 : 0  
-----  
addNode() error2: 추가 범위 초과  
현재 Node 개수 : 4  
[10]  
[50]  
[20]  
[30]  
-----  
현재 Node 개수 : 3  
[10]  
[20]  
[30]  
-----  
the location of node with data '30': 2  
현재 Node 개수 : 3  
-----  
계속하려면 아무 키나 누르십시오 . . .
```



# 4. 과제 – Univariate Polynomial Multiplication

- Call by Value & Call by Reference

```
#include <stdio.h>
void change(int x, int y) {
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    printf("변환 중 : %d, %d \n", x, y);
}

void main() {
    int x = 100, y = 200;
    printf("변환 전 : %d, %d \n", x, y);
    change(x, y);
    printf("변환 후 : %d, %d \n", x, y);
}
```

```
변환 전 :100, 200
변환 중 :200, 100
변환 후 :100, 200
```

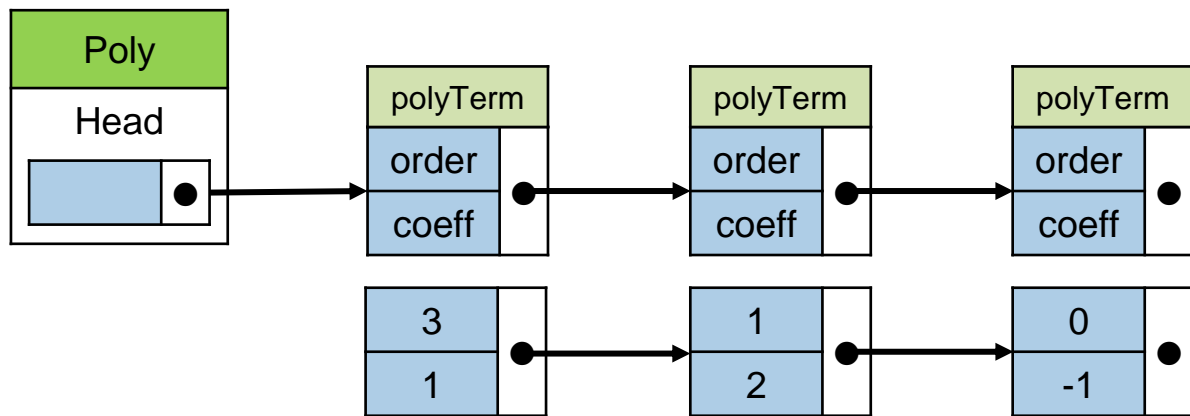
```
#include <stdio.h>
void change(int *x, int *y) {
    int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
    printf("변환 중 : %d, %d \n", *x, *y);
}

void main() {
    int x = 100, y = 200;
    printf("변환 전 : %d, %d \n", x, y);
    change(&x, &y);
    printf("변환 후 : %d, %d \n", x, y);
}
```

```
변환 전 :100, 200
변환 중 :200, 100
변환 후 :200, 100
```

# 4. 과제 – Univariate Polynomial Multiplication

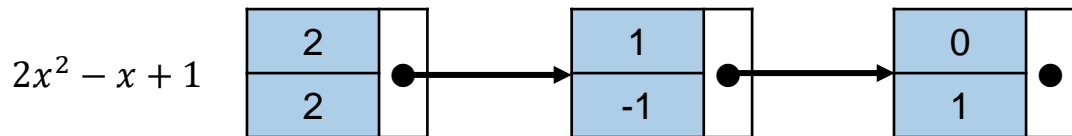
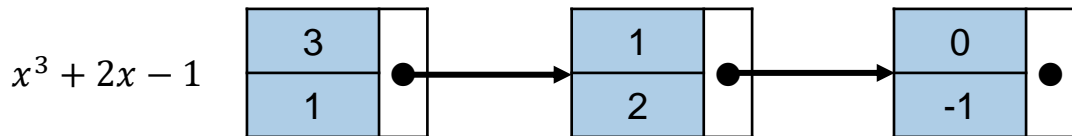
- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



$$x^3 + 2x - 1$$

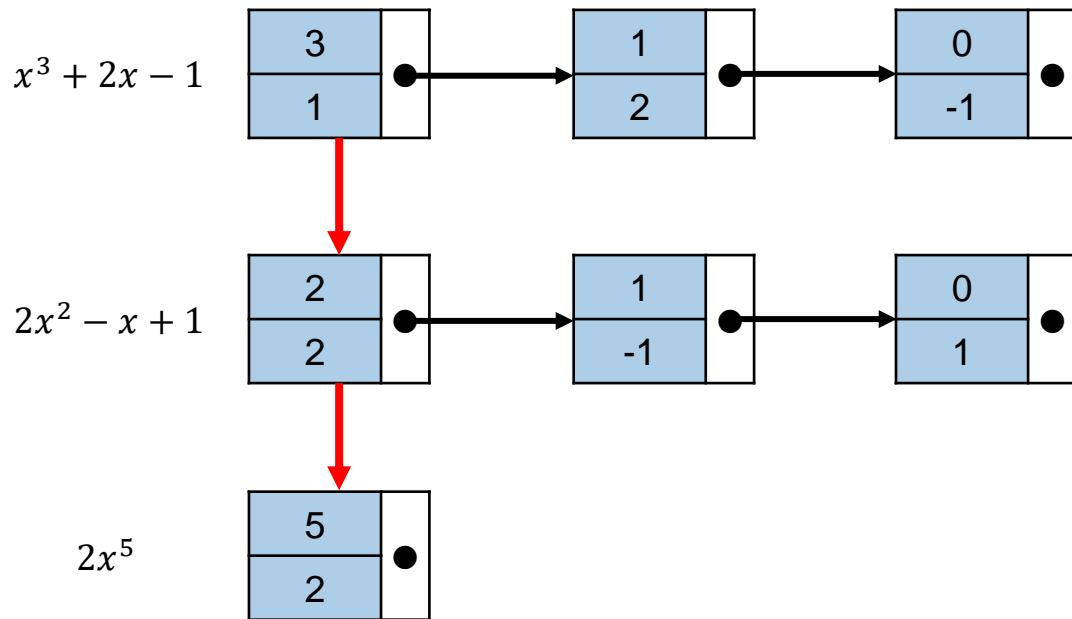
# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



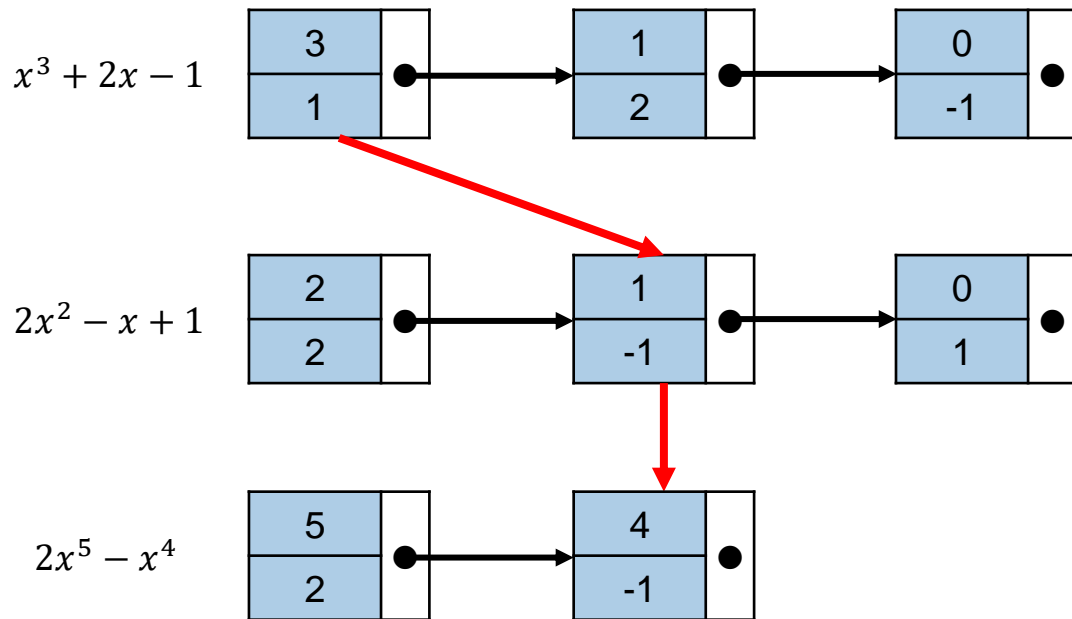
# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



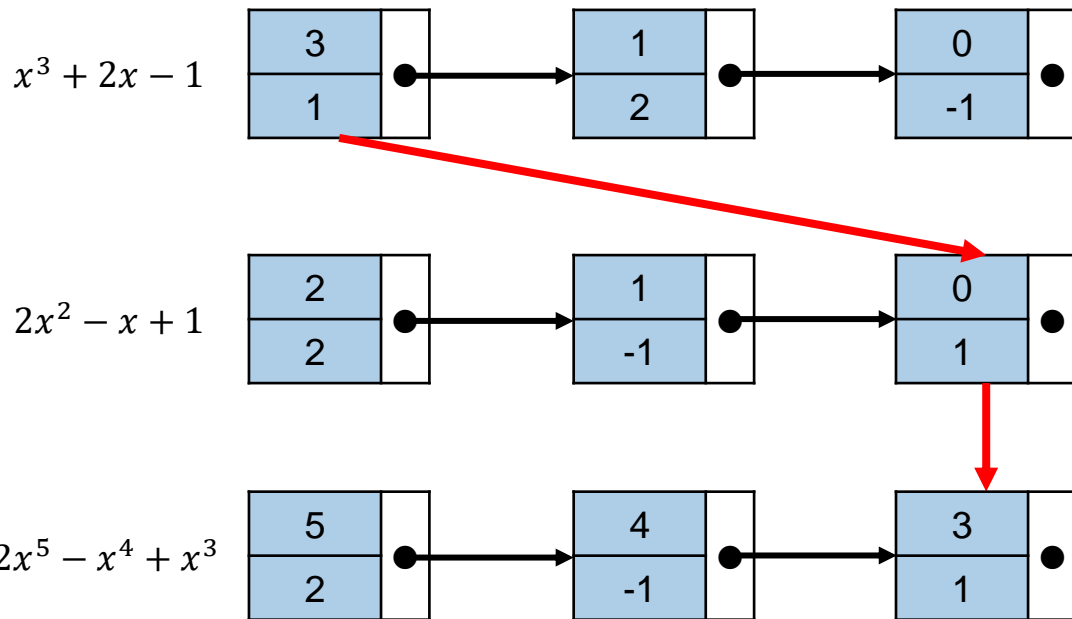
# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



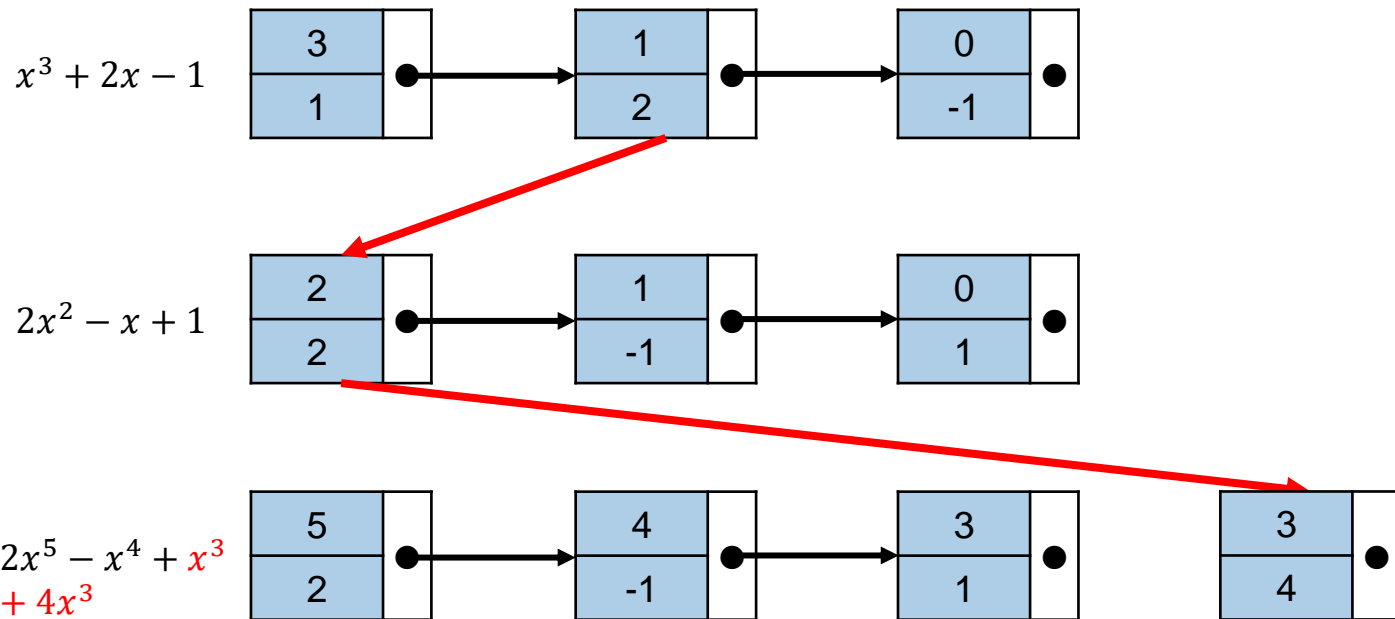
# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



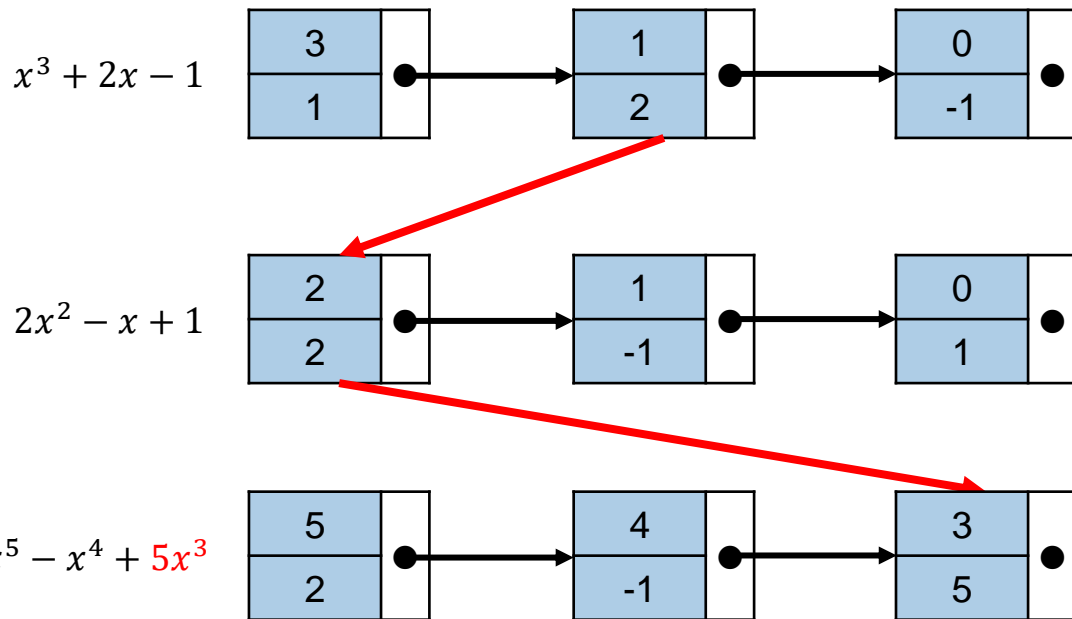
# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



# 4. 과제 – Univariate Polynomial Multiplication

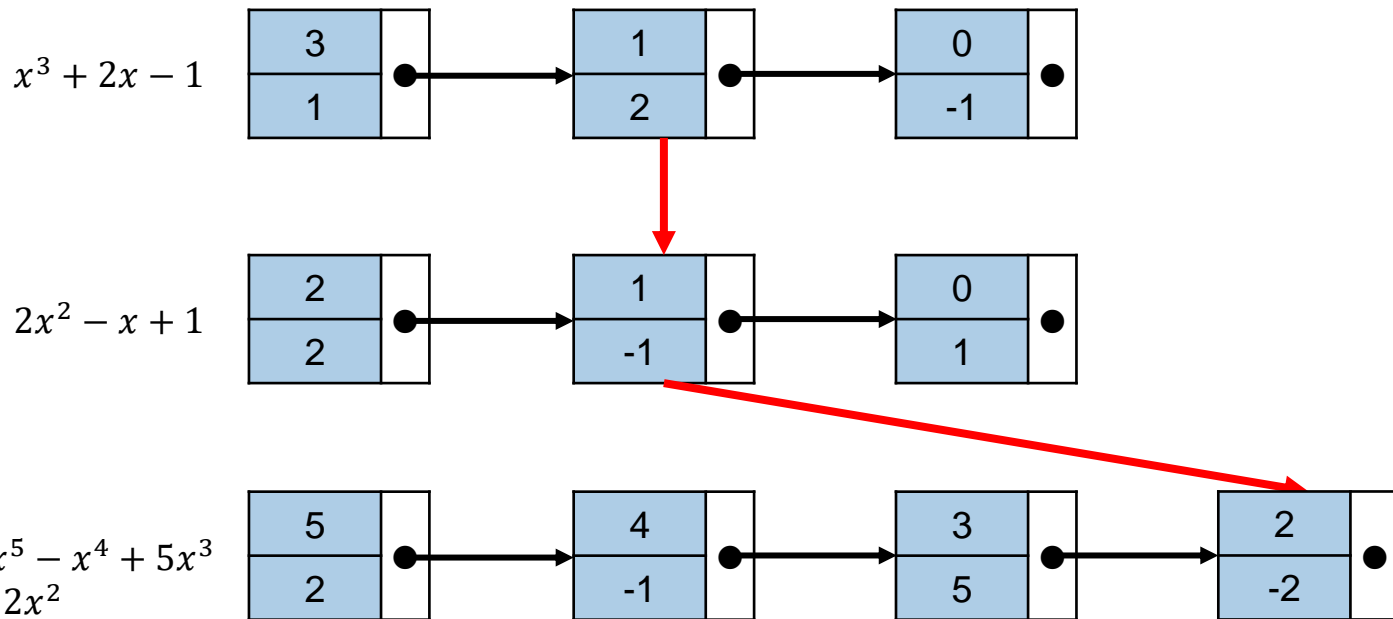
- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제





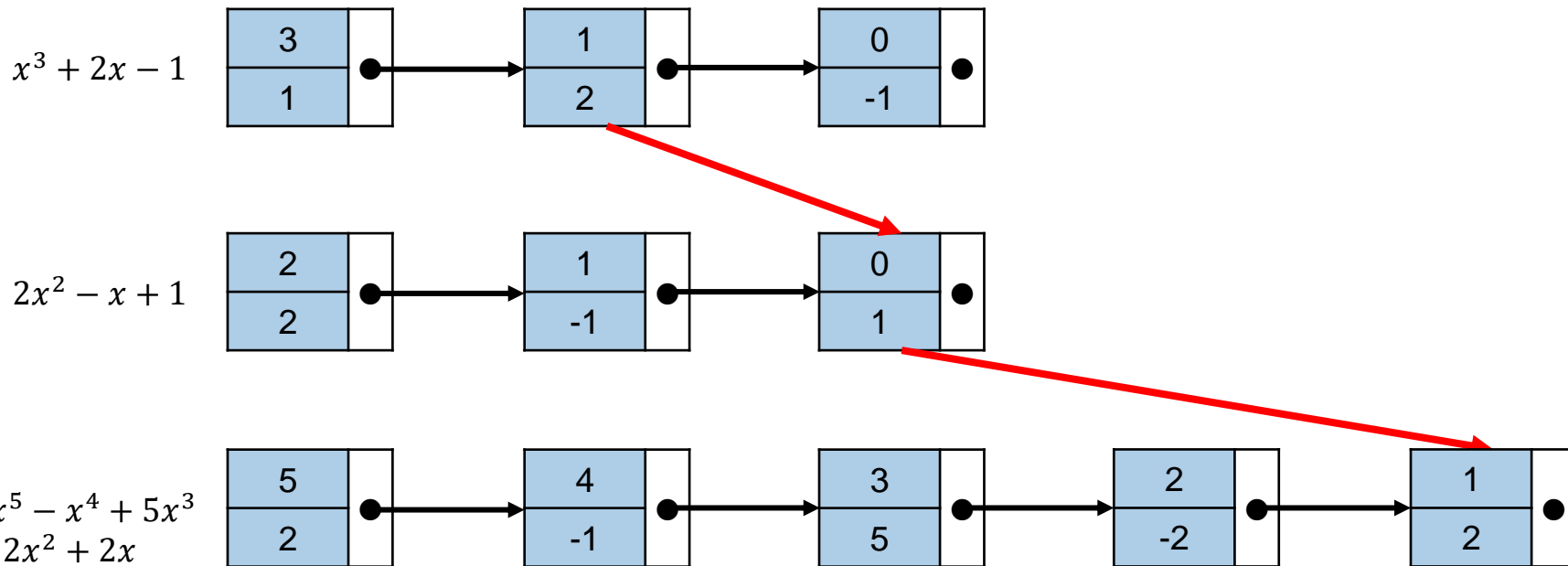
# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



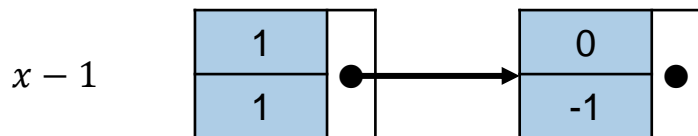
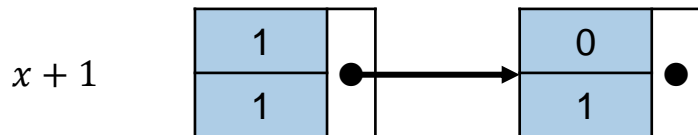
# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



# 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제



과제 하면서 직접 손으로 풀어보세요.

## 4. 과제 – Univariate Polynomial Multiplication

- 1변수 다항식 A, B를 입력 받아  $A*B$  를 계산하여 linked list에 저장하고 출력하는 과제
- 배열 기반으로 구현된 예제 코드가 제공됨
- 주어진 코드를 Linked List를 이용한 코드로 변형
- 구현해야하는 함수
  - **addTerm** (다항식에 항을 추가하는 함수, 이미 존재하는 차수의 항이면 기존의 항에 입력 항을 더함)
  - **multiPoly** (다항식 A와 B를 곱해 다항식 C를 반환하는 함수)
  - **printPoly\_impl** (다항식의 출력문자열을 문자열 버퍼에 저장하는 함수)
  - **clearPoly** (저장된 다항식을 전부 삭제하는 함수, 실행 후 0개의 항을 가짐)

# 4. 과제 – Univariate Polynomial Multiplication

- 예제 main함수 코드 및 실행결과

```
int main() {  
    poly A, B;  
  
    addTerm(&A, 1, 1);  
    addTerm(&A, 0, 1);  
    printf("poly A: ");  
    printPoly(A);  
    printf("\n");  
  
    addTerm(&B, 1, 1);  
    addTerm(&B, 0, -1);  
    printf("poly B: ");  
    printPoly(B);  
    printf("\n");  
  
    printf("A*B: ");  
    printPoly(multiPoly(A, B));  
  
    return 0;  
}
```



Microsoft Visual Studio 디버그 콘솔

```
poly A: 1x^1+1x^0  
poly B: 1x^1-1x^0  
A*B: 1x^2-1x^0
```

# 4. 과제 – Univariate Polynomial Multiplication

- 구현조건

- 각 항의 계수는 int형의 정수 (음수, 양수 모두 고려해야함)
- 각 항의 차수는 int형의 0또는 양의 정수
- 오른쪽 코드에서 //write your code here 주석이 있는 함수의 내용만 작성


```
void clearPoly(poly* A) {  
    //write your code here.  
}  
  
void printPoly_impl(poly A, char* buffer) {  
    //write your code here.  
}  
  
void printPoly(poly A) {  
    char buffer[BUFSIZE] = "";  
    printPoly_impl(A, buffer);  
    printf(buffer);  
}  
  
void addTerm(poly* A, int exp, int coeff) {  
    //write your code here.  
}  
  
poly multiPoly(poly A, poly B) {  
    //write your code here.  
}
```

polynomial\_mul.c

# 4. 과제 – Univariate Polynomial Multiplication

- **printPoly()함수의 출력형식**

- 개별 항은 (계수) $x^{(차수)}$ 의 형식으로 출력함 (예:  $4x^3$ )
  - 차수가 0인 경우에도 생략하지 않고  $4x^0$ 과 같은 방식으로 표기 해야함
- 항이 여러 개인 경우 개별항을 '+'로 연결하여 출력함 (예:  $1x^2+1x^0$ )
  - 항과 항을 연결하여 출력할 때, 뒤에 붙는 항의 계수가 음수이면 '-'로 연결하여 출력(아래의 이미지 참조)
- **차수가 큰 항부터 출력**
- 특정 차수의 항의 계수가 0인 경우 해당 항은 출력하지 않음
- 다항식에 항이 존재하지 않는 경우 0을 출력함
- printPoly()에서 줄바꿈은 따로 하지 않음

 Microsoft Visual Studio 디버그 콘솔

```
1x^1+1x^0  
1x^1-1x^0  
1x^2-1x^0
```

# 감사합니다.

과제 제출 기한: 2025년 4월 2일 23:59분 (LMS 제출 시간 기준)

제출형식:

- 소스코드의 이름을 `polynomial_mul.c`로 작성하고 LMS 과제 탭에 제출
- 과제 제출 탭은 추후 생성 예정

궁금한 것이 생기면 언제든지 질문하시면 됩니다 ☺

- 공업센터본관 304호로 방문하시거나
- [jeongiun@hanyang.ac.kr](mailto:jeongiun@hanyang.ac.kr)나 [speedpaul@hanyang.ac.kr](mailto:speedpaul@hanyang.ac.kr)로 연락 바랍니다.
- 담당교교: 이범기, 이범기