프로그래밍 연습

실습 week10

과제 풀이

실습

실습1 동적할당을 이용한 반 평균 구하기 프로그램

```
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
	int class[3];
	int *temp;
	int i=0, j=0, k=0;
	int number, score, total, avg;

	for(i=0; i<3; i++)
	{
		total=0;
		avg=0;
		scanf("%d",&number);
		temp=calloc(number,sizeof(int));
		for( j=0; j<number;j++)
		{
			scanf("%d",&score);
			temp[j]=score;
		}

		for(k=0;k<number; k++)
		{
			total+=temp[k];
		}
		avg=total/number;
		class[i]=avg;
		free(temp);
	}
	printf("%d %d %d ₩n",class[0],
class[1], class[2]);

	return 0;
}
```

실습1

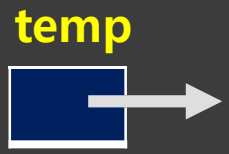
➢ 코드분석

```
int class[3];
```

- 크기가 3인 정수형 배열 class 선언 (class[0]~[2] 각 반 평균)

```
int *temp;
int i=0, j=0, k=0;
int number, score, total, avg;
```

- 배열 동적할당을 하기 위한 포인터 temp 선언
- 각 변수 i ~ avg 선언




```
for(i=0; i<3; i++)
{ …
}
```

- 각 반의 평균을 구하기 위해 반복문 실행 (3번)

```
total=0;
avg=0;
```

실습1

➢ 코드분석

```
scanf("%d",&number);

temp=calloc(number,sizeof(int));
```

- 반 학생수를 입력 받은 후, 해당 수 만큼 동적할당
- Ex) number 가 2일 경우


```
for( j=0; j<number;j++)
{
      scanf("%d",&score);
      temp[j]=score;
}
```

- 학생의 수(2) 만큼 score에 값을 입력 받아, 동적할당 배열에 저장
- Ex) 70 90 을 입력 받았을 경우

실습1

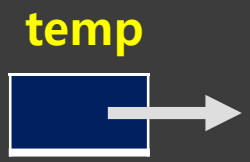
➢ 코드분석

```
for(k=0;k<number; k++)
{
      total+=temp[k];
}
avg=total/number;
class[i]=avg;
```

- 해당 반에 저장된 학생들의 평균을 구해 avg변수에 저장하고, class 배열에 저장한다.


```
free(temp);
```

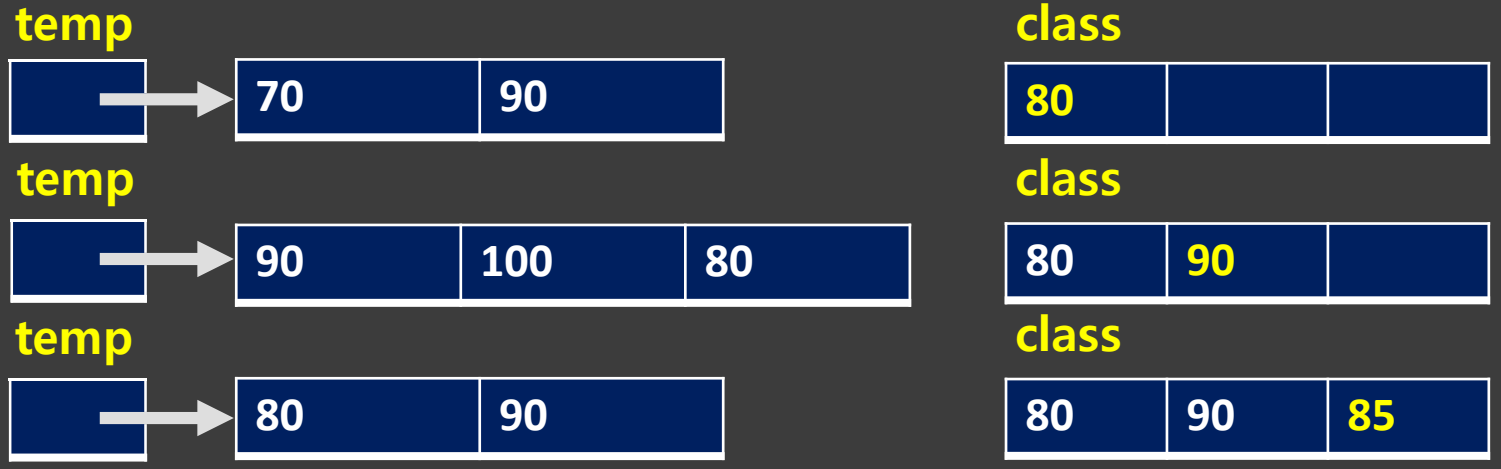
- 동적으로 할당했던 temp(메모리 공간)를 반환해준다.



➢ 코드분석

```
for(i=0; i<3; i++)
{
…
}
```

- 1) Number: 2, Score: 70, 90 2) Number: 3, Score: 90, 100, 80
  3) Number: 2, Score: 80, 85



<실행화면>

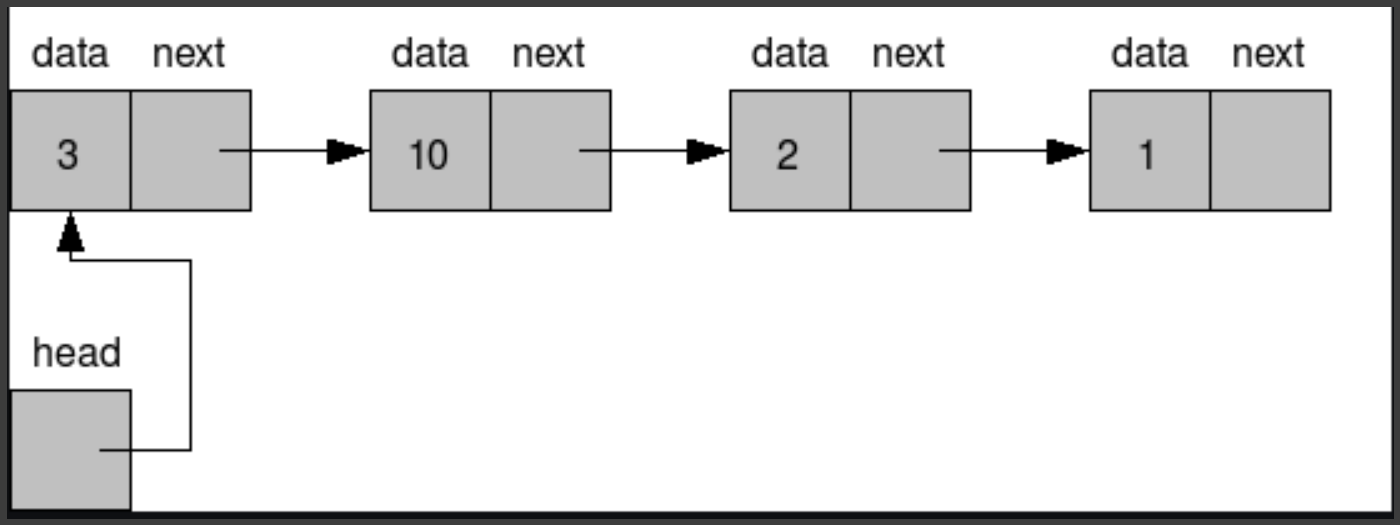
```
./ex9-1
2 70 90
3 90 100 80
2 80 90

80 90 85
```

실습 2 Singly linked list

가장 기본적인 자료구조인 singly linked list와 각종 함수 append, insert, delete 구현해봅시다.

ETL에서 skeleton code를 먼저 copy & paste 하시기 바랍니다.



실습 2 선언 부분

```
 5   #include <stdio.h>
 6   #include <stdlib.h>
 7
 8 ⊟ typedef struct __node{
 9       int data;
10       struct __node* next;
11   }node;
12
13 ⊟ typedef struct __list{
14       node* head;
15       int cnt;
16   }list;
17
18   void clear_list(list*);
19   void append_node(list*);
20   void insert_node(list*);
21   void delete_node(list*);
22   void print_list(list*);
23   void reverse_list(); //hw1
24   void sort_list();    //hw2
```

node라는 struct를 선언하는데 이는 data와 다음 노드를 가리키는 pointer 변수가 있습니다.
list struct는 linked list의 시작인 head와 list에 원소가 몇 개 있는지 counter 값을 기록합니다.

실습 2 main

```
26  int main(){
27      printf("\033[2J\033[H"); //clear screen
28      printf("\t**week10 practice**\n");
29      /* init a list */
30      list* L = (list*)malloc(sizeof(list));
31      if(!L) printf("Failed to Init.\n");
32      L->head = NULL;
33      L->cnt = 0;
```

27 : 보기 좋은 UI를 위해 콘솔창을 clear 합니다.
30 : list 구조체 L을 선언 후 memory를 dynamic allocation 합니다.
31 : 메모리가 가득 찼거나 다른 에러로 인해 allocation에 실패하면 실패를 출력합니다.
fprintf(stderr, “Failed to Init.₩n”);
위가 더 정확하나 아직 배우지 않은 부분이라 그냥 printf문으로 대체했습니다.

실습 2 main

```
34      while(1){
35        printf("a : append   i : insert  d : delete\nr :
          reverse  s : sort    p : print\nq : quit\n");
36        printf("Press key : ");
37        char c = getchar();
38        getchar(); // remove '\n'
39        printf("\033[2J\033[H"); //clear screen
40        switch(c){
41          case 'a' : append_node(L);  break;
42          case 'i' : insert_node(L);  break;
43          case 'd' : delete_node(L);  break;
44          case 'r' : reverse_list(L); break;
45          case 's' : sort_list(L);    break;
46          case 'p' : print_list(L);   break;
47          case 'q' : clear_list(L);   return 0;
48          default : printf("Invalid Key\n");
49        }
50      }
```

사용자의 입력에 따라서 함수를 호출합니다.
여기서 포인트는 getchar();로 개행문자(₩n) 지우기, default 처리 입니다.

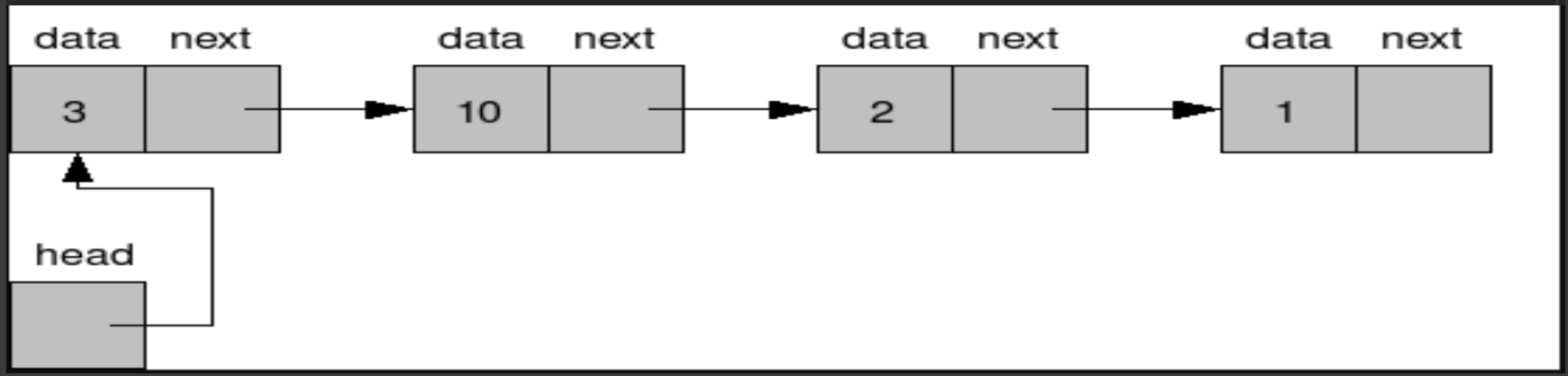
실습 2 append

```
63 ⊟void append_node(list* L){
64      node* N = (node*)malloc(sizeof(node));
65 ⊟    if(!N){
66        printf("Failed to create a node\n");
67        return;
68      }
69      int n;
70      printf("Data : ");
71      scanf("%d", &n);
72      getchar(); // remove '\n'
```

64 : 새로운 node N을 선언 후 N을 위한 memory를 allocation 합니다.
allocation이 성공하면 data를 입력 받습니다.
그 외는 위에 설명과 중복되니 생략합니다.

실습 2 append

```
73      N->data = n;
74      N->next = L->head;
75      L->head = N;
76      L->cnt++;
77      printf("\033[2J\033[H"); //clear screen
78      printf("\t Append succeeded\n");
```



73 : 입력 받은 n을 새로운 노드 N의 data 값으로 넣어줍니다.
여기서 N은 (node*) pointer이기 때문에 ->로 접근해줍니다.
~~N.data~~ N->data
새로운 node N의 next는 head를 가리키고, head는 새로운 node N을 가리킵니다.

실습 2 delete

```
116   void delete_node(list* L){
117     if(L->cnt == 0){
118       printf("Empty\n");
119       return;
120     }
121     int idx;
122     node* curr = L->head;
123     node* prev = NULL;
124     printf("Index(0~) : ");
125     scanf("%d", &idx);
126     getchar(); // remove '\n'
127     printf("\033[2J\033[H"); //clear screen
128     if(idx > L->cnt-1 || idx < 0){
129       printf("Invalid Index\n");
130       return;
131     }
```

117 : List의 element가 0이라면 삭제할 노드가 없기 때문에 바로 return 합니다.

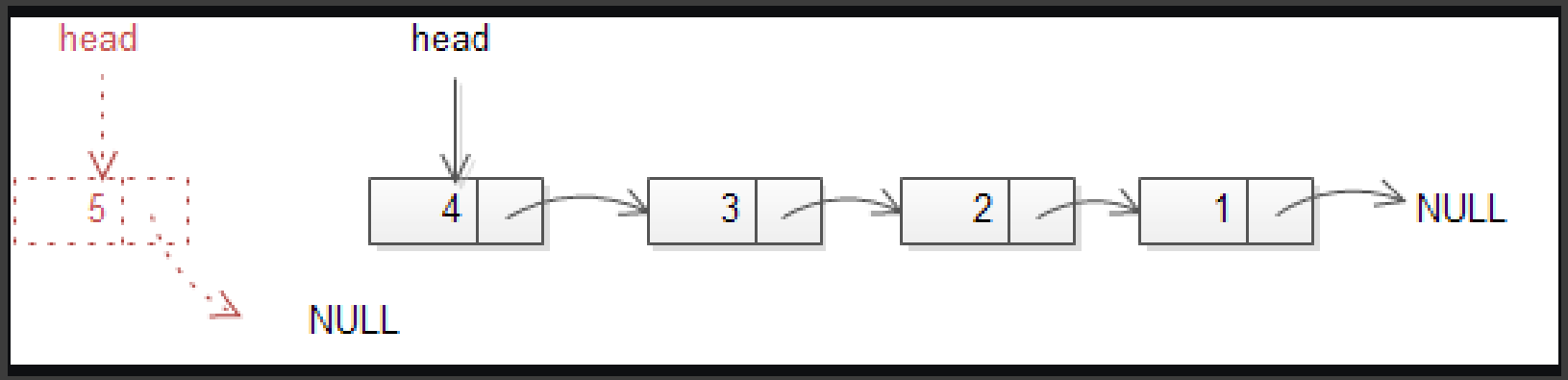
curr, prev pointer는 node delete를 위해 필요한 변수입니다.

그 후 index값이 valid 한지 검사합니다.

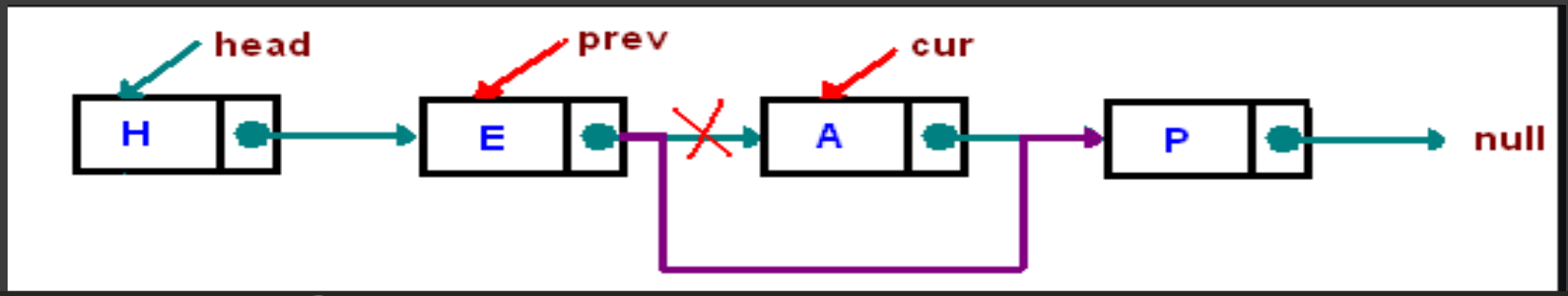
실습 2 delete

```
132     else if(idx == 0){
133       L->head = L->head->next;
134       free(curr);
135     }
```

만약 지워야 할 node의 index가 0이라면 head가 다음 노드를 가리키게 하고 기존에 가리키던 node를 지우면 끝납니다.



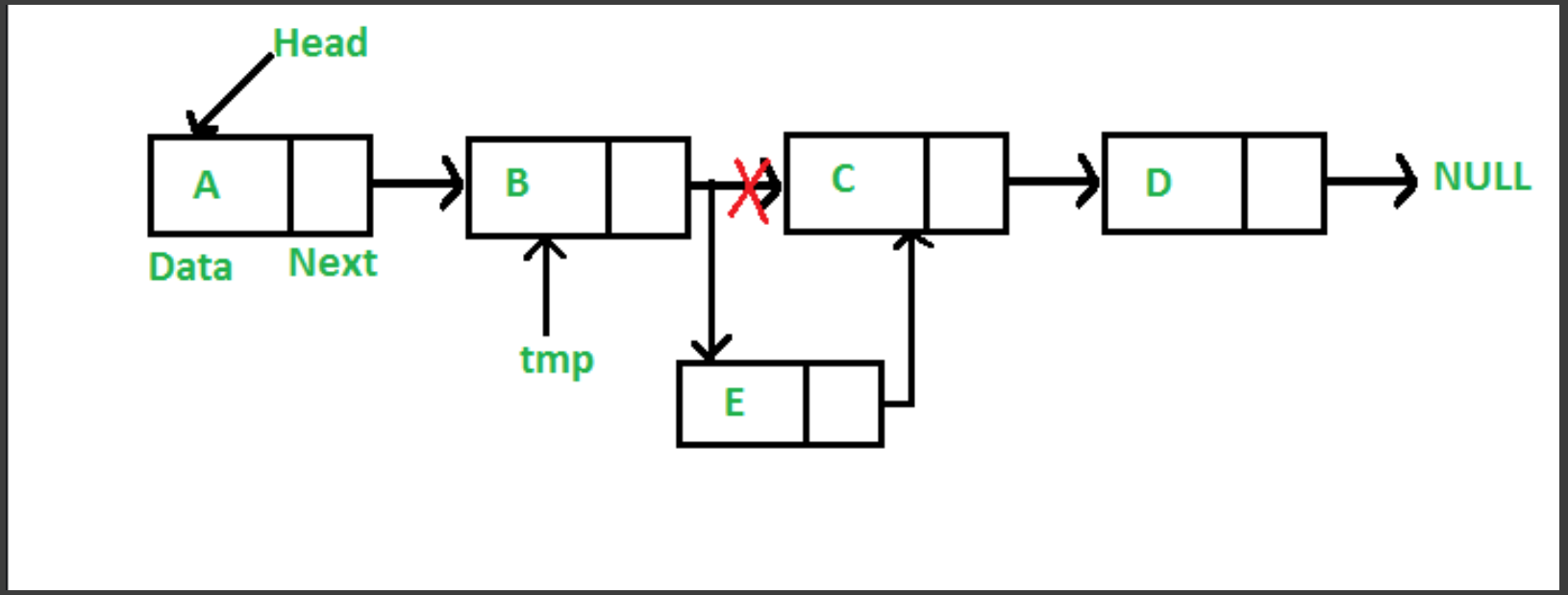
실습 2 delete




```
136     else{
137       while(idx--){
138         prev = curr;
139         curr = curr->next;
140       }
141       prev->next = curr->next;
142       free(curr);
143     }
144     L->cnt--;
145     printf("\t Delete succeeded\n");
146   }
```

만약 지워야 할 node의 index가 0 이 아니라면 idx 만큼 prev와, curr 포인터를 이동시킵니다.
그 후 prev가 가리키는 노드를 curr가 가리키는 노드로 정하고 curr을 지워줍니다.
삭제가 끝나면 counter 값을 하나 감소합니다.

실습 2 insert



그러면 원하는 위치로의 insert는 어떻게 할까요?
여기서 배열과 비교해서 list의 강점이 나옵니다.

직접 구현해보세요.
제한시간 12분

실습 2 print

```
148   void print_list(list* L){
149     if(L->cnt == 0){
150       printf("Empty\n");
151       return;
152     }
153     node* t = L->head;
154     while(t){
155       printf("%d ", t->data);
156       t = t->next;
157     }
158     printf("\n");
159   }
```

print_list 함수에서는 만약 counter가 0이라면 empty를 출력하고
아니라면 linked list를 traverse하며 값을 출력합니다.
이 때 중요한 점은 L->head를 직접 쓰는게 아니라 임시변수를 이용해
traverse를 하는 겁니다.
head 값이 바뀔 수 있기 때문입니다.

실습 2 clear

```
54 ⊟void clear_list(list* L){
55 ⊟   while(L->head){
56        node* tmp = L->head;
57        L->head = L->head->next;
58        free(tmp);
59     }
60     free(L);
61  }
```

함수가 종료 될 때는 그동안 allocation 했던 자원들을 다 반환해줘야 합니다.
C언어는 자동으로 자원들을 반환해주는 garbage collection이 없기 때문에 직접 구현해야 합니다.

과제

과제 (파일명 : hw.c)

문제.

수업 시간에 사용한 skeleton code의
reverse_list() //hw1
sort_list() //hw2
함수를 구현하세요.

reverse_list 함수는 list에 있는 node들의 순서를 역순으로 바꿉니다.

```
before
1 2 3 4 5
```

```
after
5 4 3 2 1
```

sort_list 함수는 node를 오름차순으로 정렬합니다.
여기서 sorting은 아무거나 쓰셔도 무방합니다. (bubble, merge.. etc)

```
before
3 2 4 5 1
```

```
after
1 2 3 4 5
```