자료구조 및 알고리즘 프로젝트

1. 중간고사 대체 과제는 이중 연결 리스트와 관련된 아래의 6개 함수를 구현하는 과제.

구현 함수
int cal_length(DLlistType* dlist);
DLlistNode* node_position(int pos, DLlistType* dlist);
DLlistNode* delete_pos(int pos, DLlistType* dlist);
void insert_node_pos(int pos, int entrance_year, char name[MAX_SIZE], float GPA, DLlistType* dlist);
void insert_inc_entrance_year(int entrance_year, char name[MAX_SIZE], float GPA, DLlistType* dlist);
void insert_dec_GPA(int number, char name[MAX_SIZE], float GPA, DLlistType* dlist);

2. “project_main_학과_학번_이름.cpp” 파일로 6개 함수를 구현하고 구현한 함수 내용을 한글 파일(중간고사_대체_과제_학과_학번_이름.hwp)에도 복사를 합니다. 총 2개의 파일을 제출.

3. 구현을 위한 구체적인 요구 사항은 다음과 같습니다.
. 구현할 함수 안에서 필요한 지역 변수와 지역 포인터를 선언하여 사용 가능.
. 구현할 함수 안에서 사용(호출)할 추가적인 함수 구현 가능.
. main 함수의 “st_array” 배열에 있는 학생 정보 및 학생 수는 변경 가능하다는 가정하고 코딩.
. 구현한 코드에 대한 설명을 주석으로 작성.
. 전역 변수(전역 포인터) 사용 금지.
. 주어진 구조체 이외의 추가적인 구조체 선언 금지.
. main 함수 수정 금지.
. 6개 구현 함수의 함수 원형 수정 금지.

1. cal_length 함수: 이중 연결 리스트의 노드들의 개수를 계산하여 반환하는 함수.

```
int cal_length(DLlistType* dlist) {
        DLlistNode* x = dlist->head; //돌아다닐 노드 x
        int l = 1; //길이를 저장할 변수
        if (dlist->head == NULL)
                return 0; // 리스트가 비어있다면 길이를 0으로 반환한다.(임의로 정하
   였습니다)
        while (x->rlink != NULL)  //x가 없을(비어있을) 때 까지
        {
                l++; //길이+1
                x = x->rlink; //다음으로 이동
        }
        return l; // 길이 l을 반환
}
```

2. node_position 함수: 이중 연결 리스트에서 position 위치의 노드 주소를 반환하는 함수.
* 리스트가 비어 있으면, "리스트 공백" 이라는 메시지를 출력하고 그리고 NULL을 반환.
* 리스트가 비어 있지 않으면, position이 사용 가능한 위치 여부를 확인하고, 만약 position이 사용 가능한 위치가 아니면, "position 오류: position: _ ~ _ 선택" 이라는 오류 메시지를 출력하고 NULL을 반환. 만약, position이 사용 가능한 위치이면, position에 해당하는 노드의 주소를 반환.

```
DLlistNode* node_position(int pos, DLlistType* dlist){
        DLlistNode* x = dlist->head; //리스트를 돌아다닐 x
        int n = 1; //x의 위치를 나타냄
        if (dlist->head == NULL) // 비어있는 경우
        {
                printf("\n리스트 공백\n");
                return NULL; // NULL 반환
        }
        else if (pos > cal_length(dlist)) // 찾을 position이 리스트이 길이보다 큰 경
   우
        {
                printf("\nposition 오류: position: 1~%d 선택\n", cal_length(dlist));
                return NULL; // NULL 반환
        }
        else
        {
                while (n != pos) //x를 pos번째 노드로 이동시키기
                {
                        x = x->rlink;
                        n++;
                }
                return x; //그 때의 x를 반환
        }
}
```

3. delete_pos 함수: 이중 연결 리스트에서 position에 해당하는 노드를 삭제하는 함수.
* 리스트가 비어 있으면, "공백 삭제 오류" 이라는 메시지를 출력하고 NULL을 반환.
* 리스트가 비어 있지 않으면, position이 사용 가능한 위치 여부를 확인하고, 만약 position이 사용 가능한 위치가 아니면, "position 오류: position: _ ~ _ 선택" 이라는 오류 메시지를 출력하고 NULL을 반환. 만약, position이 사용 가능한 위치이면, 노드를 삭제하고 삭제된 노드의 주소 반환.

```
DLlistNode* delete_pos(int pos, DLlistType* dlist){
	DLlistNode* x = dlist->head;
	DLlistNode* y = NULL;
	int n = 1;
	if (dlist->head == NULL) //리스트가 비어있는 경우
	{
		printf("\n공백 삭제 오류\n");
		return NULL;
	}
	else if (pos > cal_length(dlist)) // 삭제할 position이 리스트이 길이보다 큰 경우
	{
		printf("\nposition 오류: position: 1~%d 선택\n", cal_length(dlist));
		return NULL; // NULL 반환
	}
	else if (pos == 1 && (cal_length(dlist) == 1)) // 길이가 1인(요소가 하나인 리스트) 경우
	{
		DLlistNode* z = x;
		x = y;
		return z;
	}
	else if (pos == 1) //첫 노드를 삭제하는 경우 (길이 2 이상)
	{
		dlist->head = dlist->head->rlink; // 헤더노드를 기존 헤더노드 다음 노드로 지정함으로써 리스트에서의 삭제와 같은 효과
		return x; //제거한 기존 헤더노드 반환
	}
	else if (pos == cal_length(dlist)) //마지막 노드 제거
	{
		DLlistNode* z = NULL; // x를 임시로 받을 z
		while (n != pos-1) //x를 맨 끝 이전 노드로 이동
		{
```

```
                        x = x->rlink;
                        n++;
                }
                z = x->rlink; //z에 position위치의 노드를 지정
                y = z; //입력받은 position의 노드를 y에 저장
                x->rlink = NULL; // x(pos이전 노드)와 pos위치에 있는 노드와의 연결을 끊음
                z->llink = NULL; //z의 llink를 끊음으로써 pos위치에 있는 노드와 pos이전 노드와의 연결을 끊음

                return y;
        }
        else // 중간 노드 삭제
        {
                DLlistNode* z = NULL;
                while (n != pos - 1) //x를 position 이전 노드로 이동
                {
                        x = x->rlink;
                        n++;
                }
                z = x->rlink; //z에 postion위치의 노드를 지정
                y = z; //y에 z 저장
                z = z->rlink; //z에 position위치 다음 노드를 지정
                x->rlink = z; //x(postion이전 노드)가 z(position다음 노드)를 가리키게
                z->llink = x; //마찬가지로 z(position다음 노드)가 x(position 이전 노드)를 가리키게 함으로써 position 위치의 노드를 리스트에서 삭제한 것과 같은 효과.
                return  y;
        }
}
```

4. insert_node_pos 함수: 이중 연결 리스트에서 지정한 position에 새로운 노드 삽입 함수.
* 만약 position이 사용 가능한 위치가 아니면, "position 오류: position: _ ~ _ 선택" 이라는 오류 메시지를 출력하고 함수 종료(프로그램 종료가 아님).
* 만약 position이 사용 가능한 위치이면, 새로운 노드를 생성한 후에,
만약 리스트가 비어 있으면, 리스트의 맨 처음 노드로 삽입. 만약 리스트가 비어 있지 않으면, position에 노드 삽입.

```
void insert_node_pos(int pos, int entrance_year, char name[MAX_SIZE], float GPA, DLlistType* dlist){
	DLlistNode* x = dlist->head; // 돌아다닐 노드 x
	DLlistNode* y = create_node(entrance_year, name, GPA, NULL, NULL); // 삽입될 위치에 들어갈 노드 y
	int n = 1;
	if (dlist->head == NULL) //리스트가 비어있는 경우 => pos에 상관없이 리스트의 맨 처음 노드(position = 1)으로 삽입
	{
		dlist->head = y;// y를 리스트의 헤더노드로 설정
	}
	else if (pos > cal_length(dlist)+1) // 삽입할 수 없는 위치를 입력받은 경우
	{
		printf("\nposition 오류: position: 1~%d 선택\n", cal_length(dlist)+1);
	}
	else if (pos == 1) //리스트가 비어있지 않으면서 position이 1인 경우 : 리스트의 처음 노드 위치에 삽입하는 경우
	{
		dlist->head = y; // y를 헤더노드로 만듬
		x->llink = y; // 기존의 헤더노드인 x에서 x의 이전 노드로 y를 지정
		y->rlink = x; //y의 다음 노드로 x를 지정
	}
	else if (pos == (cal_length(dlist)+1)) //리스트의 가장 마지막에 삽입하는 경우
	{

		while (n != pos-1) //리스트이 맨 끝으로 이동 : pos=1+length이므로 pos-1이 리스트의 맨 끝
		{
			x = x->rlink;
			n++;
		}
		y->llink = x; // 삽입될 노드의 이전 노드로 x(기존 맨끝노드)를 지정
		x->rlink = y; // 기존의 맨 끝 노드 x의 다음 노드로 y를 지정
	}
	else if(pos >= 2)// 리스트가 비어있지 않으면서 postion이 1보다 큰 경우 : 중간에 삽입하는 경우
	{
		DLlistNode* z = x; //z라는 새로운 변수에 x를 저장해놓음(후에 x를 한 번 더 이동시키기 위해서)
```

```
                while (n != pos-1) //x를 pos번째 노드 이전 노드로 이동
                {
                        x = x->rlink;
                        n++;
                }
                z = x->rlink; //z에는 기존 pos번째 노드를 지정
                x->rlink = y; // x(기존의 pos-1번째 노드) 다음 노드로 y를 지정
                y->llink = x; // y의 이전 노드로 x를 지정
                y->rlink = z; // y의 다음노드로 z(기존 pos번째 노드)를 지정
                z->llink = y; // z의 이전 노드로 y를 지정
        }
    }
```

5. insert_inc_entrance_year 함수: 이중 연결 리스트에 새로운 노드를 삽입하는 함수이며 리스트의 노드의 입학년도가 증가하는 순서가 되도록 삽입하는 함수.
* 만약 리스트가 비어 있으면, 새롭게 생성된 노드를 리스트의 맨 처음 노드로 삽입. 만약 리스트가 비어 있지 않으면, 새롭게 생성된 노드의 입학년도와 이중 연결 리스트의 노드의 입학년도를 비교하여 입학년도가 오름차순이 되도록 삽입 위치를 찾아서 삽입.
(2012 -> 2015 -> 2017)

```
void insert_inc_entrance_year(int entrance_year, char name[MAX_SIZE], float GPA,
DLlistType* dlist){
        DLlistNode* x = dlist->head;
        DLlistNode* n = create_node(entrance_year, name, GPA, NULL, NULL);
        DLlistNode* z = NULL;
        int t = 1; // 케이스들을 구분지을 변수
        if (dlist->head == NULL) //리스트가 비어있던 경우
        {
                dlist->head = n; //첫 노드 지정
        }
        else
        {
                if (dlist->head->st_entrance_year > entrance_year) //새로운
   entrance_year가 기존 헤드 노드의 entrance_year 보다 작은 경우
                {
                        dlist->head = n;
                        x->llink = n;
                        n->rlink = x;
                }//새로운 노드를 헤더노드로 지정
                else if (dlist->head->st_entrance_year < entrance_year) //새로운
```

```
entrance_year가 기존 헤드 노드의 entrance_year 보다 큰 경우
            {
                    if (cal_length(dlist) == 1) // 리스트의 길이가 1인 경우
                    {
                            x->rlink = n;
                            n->llink = x; // 리스트 끝에 삽입하기
                    }
                    else if(cal_length(dlist) > 1) // 리스트의 길이가 1보다 큰 경
우
                    {
                            while (x->st_entrance_year < entrance_year) //
새로운 entrance_year가 노드 중 entrance_year보다 크면 계속해서 반복
                            {
                                    if (x->rlink == NULL) //다음 노드가 없는
상황 : 리스트의 노드들의 모든 entrance보다 삽입할 entrance가 더 큰 경우
                                    {
                                            x->rlink = n;
                                            n->llink = x; //리스트 마지막에 삽
입
                                            t++; // 아래(while문 밖)의 구문을
실행하지 않게 하기 위한 작업
                                            break; // 반복문을 나가기
                                    }
                                    x = x->rlink; // 다음 노드로 이동
                            }
                            if (t == 1) // 중간에 삽입되는 경우
                            {
                                    z = x; //z에 x저장
                                    z = z->llink; // 한칸 앞으로 이동
                                    n->llink = z; // n의 이전 노드로 z를 지정
                                    z->rlink = n; // z의 다음 노드로 n을 지정
-> 삽입노드(n)의 이전 노드와의 연결 완료
                                    n->rlink = x; // n의 다음 노드로 x 지정
                                    x->llink = n; // x의 이전 노드로 n 지정 ->
삽입노드의 다음 노드와의 연결 완료
                            }
                    }
            }
    }
}
```

6. insert_dec_GPA 함수: 이중 연결 리스트에 새로운 노드를 삽입하는 함수이며 리스트의 노드의 학점이 감소하는 순서가 되도록 삽입하는 함수.
* 만약 리스트가 비어 있으면, 새롭게 생성된 노드를 리스트의 맨 처음 노드로 삽입. 만약 리스트가 비어 있지 않으면, 새롭게 생성된 노드의 학점과 이중 연결 리스트의 노드의 학점과 비교를 하여 학점이 내림차순이 되도록 삽입 위치를 찾아서 삽입.
(3.3 ->3.2 -> 3.1)

```
void insert_dec_GPA(int entrance_year, char name[MAX_SIZE], float GPA, DLlistType* dlist){
	DLlistNode* x = dlist->head;
	DLlistNode* n = create_node(entrance_year, name, GPA, NULL, NULL);
	DLlistNode* z = NULL;
	int t = 1; // 케이스들을 구분지을 변수
	if (dlist->head == NULL) //리스트가 비어있던 경우
	{
		dlist->head = n; //첫 노드 지정
	}
	else
	{
		if (dlist->head->st_GPA < GPA) //새 GPA가 기존 헤드노드의 GPA보다 큰 경우
		{
			dlist->head = n;
			x->llink = n;
			n->rlink = x;
		}//새로운 노드를 헤더노드로 지정
		else if (dlist->head->st_GPA > GPA) //새 GPA가 기존 헤더가 가리키는 노드의 GPA보다 작은 경우
		{
			if (cal_length(dlist) == 1) // 리스트의 길이가 1인 경우
			{
				x->rlink = n;
				n->llink = x;
			}
			else if (cal_length(dlist) > 1) // 리스트의 길이가 1보다 큰 경우
			{
				while (x->st_GPA > GPA) // 새 GPA가 리스트 노드의 GPA보다 작으면 계속해서 반복
				{
					if (x->rlink == NULL) // 다음 노드가 없음 : 새 GPA가 모든 노드의 GPA 보다 작은 경우
```

```
                                        {
                                                x->rlink = n;
                                                n->llink = x; //n과 x(기존 리스트
   의 마지막 노드)와 연결
                                                t++; // 아래(while문 밖)의 구문을
   실행하지 않게 하기 위한 작업
                                                break;
                                        }
                                        x = x->rlink; //이동
                                }
                                if (t == 1) // 중간에 삽입되는 경우
                                {
                                        z = x; //z에 x저장
                                        z = z->llink; // 한칸 앞으로 이동
                                        n->llink = z; // n의 이전 노드로 z를 지정
                                        z->rlink = n; // z의 다음 노드로 n을 지정
   -> 삽입노드(n)의 이전 노드와의 연결 완료
                                        n->rlink = x; // n의 다음 노드로 x 지정
                                        x->llink = n; // x의 이전 노드로 n 지정 ->
   삽입노드의 다음 노드와의 연결 완료
                                }
                        }
                }
        }
}
```

※ 주의사항

1. 과제는 반드시 본인이 작성하여 제출.
2. 카피 체크 프로그램으로 체크하며, 과제 카피가 확인되면, 관련된 모든 과제 미제출 처리.
3. 제출 기간을 넘기면 감점을 하며 제출 기한에서 1주일이 지난 후에는 과제를 받지 않음.