```
//01
//2020118008 박보경
//본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define MALLOC(p, s) \
	if( !( (p) = malloc( s ) ) ){		\
		fprintf(stderr, "Insufficient memory");\
		exit(EXIT_FAILURE);\
	}

// linked list
typedef struct listNode* listPointer;
struct listNode {
	int data;
	listPointer link;
};


void find(listPointer first, listPointer* x, int data);
void insert(listPointer* first, listPointer x, int data);
void delete(listPointer* first, listPointer trail, listPointer x);
void printList(listPointer first);

int main(void)
{
	int data;
	listPointer x, trail, curr, ffirst;
	listPointer first = NULL;


	// data input for each node
	FILE* fp=fopen("input.txt", "r");
	if (fp == NULL)
	{
		fprintf(stderr, "cannot open the file");
		exit(EXIT_FAILURE);
	}

	fscanf_s(fp, "%d", &data);
```

```c
        while (!feof(fp))
        {
                find(first, &x, data);              // find insert position
                insert(&first, x, data);            // insert data first after node x.
                fscanf_s(fp, "%d", &data);
        }
        printList(first);

        curr = ffirst = first;
        trail = NULL;

        while (curr != NULL)
        {
                if (curr->data <= 50) {
                        x = curr;
                        delete(&first, trail, x);
                        if (ffirst != first) {
                                trail = NULL;
                                curr = first;
                                ffirst = first;
                        }
                        else
                                curr = trail->link;

                }
                else
                {
                        trail = curr;
                        curr = curr->link;
                }
        }

        printf("\nAfter deleting nodes with data less than and equal to 50 \n\n");
        printList(first);

        fclose(fp);

        return 0;
}


void find(listPointer first, listPointer* x, int data)
{ /* *x is the position of insert  */
```

```
	listPointer trav = first;

	if (first == NULL)
	{ // empty list
		*x = first;
	}
	else
	{ // non-empty list
		for (; trav; trav = trav->link)
		{
			if (data > trav->data)
				*x = trav;
			else
			{
				if (first == trav)
					*x = NULL;
				break;
			}
		}
	}
}

void insert(listPointer* first, listPointer x, int data)
{ /* insert a new node with a data into the chain first after node x */
	listPointer temp;
	MALLOC(temp, sizeof(*temp));
	temp->data = data;

	if (*first == NULL)
	{ // add to empty list
		temp->link = NULL;
		*first = temp;
	}
	else
	{ // add to non-empty list

		if (x == NULL)
		{ // as a first node
			temp->link = *first;
			*first = temp;
		}
		else
		{
```

```
                    temp->link = x->link;
                    x->link = temp;
            }
        }
}

void delete(listPointer* first, listPointer trail, listPointer x)
{ /* delete x from the list, trail is the preceding node
          and *first is the front of the list */
        if (trail)
                trail->link = x->link;
        else
                *first = (*first)->link;
        free(x);
}

void printList(listPointer first)
{
        int count;
        int i = 0;
        printf("The ordered list contains: \n");
        for (count = 1; first; first = first->link, count++)
        {

                printf("%4d", first->data);
                ++i;
                if ((i % 10) == 0)
                        printf("\n");
        }
        printf("\n");
}
```