Name: Hemos ID:

Student ID (학번):

CSED-101 PROGRAMMING & PROBLEM SOLVING

Spring 2014

Final

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Total
Your Score													
Max Score	6	4	5	6	7	16	8	5	15	6	5	17	100

- Write down your name, hemos ID, and student ID.
- There are 19 pages in this final.
- Your answers must run correctly in **C programming language** without error or warning. Otherwise, your answers will be considered incorrect. For example, it is ok to put more parentheses than needed in your answer, but it will be incorrect if you put fewer parentheses than needed.
- You must write your answer on the underline => _____. Scratches outside the underline will be ignored.
- The total score is 100
- This is a 3-hour exam.

명예서약서

나는 명예로운 포스테키안으로서 다음의 Honor Code 를 지킬 것을 맹세합니다.

정직과 타인에 대한 존중이 함께하는 포스테키안의 미래는 명예롭다.

2014 년 월 일

이름 (인)

1. (6 점) 아래 프로그램의 출력결과를 쓰시오.

```
#include <stdio.h>

int main()
{
char arr[][6] = {"cs101","final","exam","is","easy"};

    printf("%s\n", arr[2]);       // (1)
    printf("%c\n", *(&arr[3][1]-1)); // (2)
    printf("%c\n", (*(arr+4))[2]);    // (3)
    printf("%c\n", ((char *)arr)[6]);// (4)
    printf("%s\n", arr[1]+1);     // (5)
    printf("%c\n", *(*(arr+2)+1));    // (6)

    return 0;
}
```

출력 결과

(1) (1 점)	exam
(2) (1 점)	i
(3) (2 점)	s
(4) (1 점)	f
(5) (1 점)	inal
(6) (1 점)	x

2. (4 점) 아래 프로그램의 출력결과를 쓰시오.

```
#include <stdio.h>

int g=0;
void func (int *n);

int main()
{
    int a=0;

    func(&a);
    func(&a);

    printf("%d\n", a);     // (1)
    printf("%d\n", g);     // (2)
    {
        int g=1;
        printf("%d\n",g);  //(3)
    }
    printf("%d\n", g);     //(4)

    return 0;
}

void func( int *n)
{
    static int s = 1;
    g = g + s;
    (*n)++;
    s++;
}
```

출력 결과

(1) (1 점)	2
(2) (1 점)	3
(3) (1 점)	1
(4) (1 점)	3

3. (5점) 다음은 순차탐색을 수행하는 프로그램이다. 배열의 원소 중 찾고자 하는 값이 있으면 그 값의 index를 출력하고, 없으면 "Not Fount."를 출력한다. 아래의 실행예제를 참고하여 빈 칸을 채우시오.

실행예제 1)

```
Input an integer that you want to fount: 10
The index of 10 is 7.
```

실행예제 2)

```
Input an integer that you want to fount: 2
Not Fount.
```

```
#include <stdio.h>
#define ARY_SIZE 10

int seq_search(int list[], int last, int target, int *pidx);

int main()
{
    int arr[ARY_SIZE] = {3, 4, 1, 7, 8, 9, 6, 10, 11, 15};
    int target, index;
    int fount;

    printf("Input an integer that you want to find: ");
    scanf("%d",&target);

    fount = seq_search(arr,ARY_SIZE-1, target, ____&index____);
    if(fount)
        printf("The index of %d is %d.\n", target, index);
    else
        printf("Not Fount.\n");

    return 0;
}

// 찾고자 하는 값(target)이 배열에 있으면 1, 아니면 0을 반환한다.
// 단, 여기서 last는 배열의 마지막 원소의 index 값이 전달된다.
int seq_search(int list[], int last, int target, int *pidx)
{
    int looker;
    int fount;

    looker = 0;
    while(looker < last && ___list[looker]!=target___)
        looker++;

    _*pidx=looker__;

     fount = __(list[looker]==target) ? 1:0__;

    return fount;
}
```

4. (6점) 아래의 프로그램은 배열의 원소 중 가장 작은 값을 찾아 그 값을 출력하는 프로그램이다. 배열 arr 에서 pmin 이 가장 작은 값을 가리키도록 빈 칸을 채우시오.

```
#include <stdio.h>
#define ARY_SIZE 10

void set_min_prt (int *arr, int size, int **pmin);

int main()
{
    int arr[ARY_SIZE] = {3, 4, 1, 7, 8, 9, 6, 10, 11, 15};
    int *pmin; // 배열의 원소 중 가장 작은 값을 가리키기 위한 변수

    set_min_ptr (arr, ARY_SIZE, ____&pmin____);

    printf("Minimum is %d,\n", ____*pmin___);

    return 0;
}


void set_min_prt (int *arr, int size, int **pmin)
{
    int i;
    ___*pmin___ = arr;

    for(i = 1; i < size; i++)
    {
       if(___**pmin___ > ____arr[i]___)
       {
          ________*pmin=&arr[i]________;
       }
    }
    return;
}
```

실행 결과:

```
Minimum is 1.
```

5. [7점] 다음은 버블정렬(Bubble Sort)과 관련된 문제이다.

5.1. (5 점) 아래의 함수 bubble_sort 는 버블정렬(Bubble sort) 알고리즘을 사용하여 배열을 오름차순으로 정렬한다. 단, 뒤에서부터 시작한다. 빈 칸을 채우시오.

```
#include <stdio.h>

void swap(int *first, int *second); //두 변수의 내용을 교환

void print_array(int list[], int last);//배열의 원소를 index 0 부터 last 까지 출력

void bubble_sort(int list[], int last) //last 는 배열의 마지막 원소의 index
{
    int cur, walk;

    for(cur = 0; cur < last; cur++)
    {
       for(walk = __last__; walk > cur; ____walk--___)
          if(__list[walk]__ < __ list[walk-1]___)
          swap(__&list[walk]__,__&list[walk-1] ___;

       printf("[step %d] : ",cur+1);
       print_array(list, last);
    } //for cur
    return;
}
```

5.2. (2 점) 위의 함수를 아래와 같이 호출하였을 때 출력 결과를 쓰시오.

```
int list[] = {56, 3, 21, 92, 47, 33};

bubble_sort (list,5);
```

	56	3	21	92	47	33
step 1	3	56	21	33	92	47
step 2	3	21	56	33	47	92
step 3	3	21	33	56	47	92
step 4	3	21	33	47	56	92
step 5	3	21	33	47	56	92

6. (16점) 다음은 한 과목의 성적 데이터 파일(텍스트 파일)로부터 시험 성적을 읽어 각 학생의 평균 점수 및 과목 평균 등을 구하여 화면에 출력하는 프로그램이다. 주석을 참조하여 다음 조건을 만족하는 프로그램을 완성하시오.

6.1. 아래처럼 실행명령어를 입력하여 프로그램을 실행시킨다.(밑줄은 사용자 입력을 말함)

```
C:₩>score.exe score.txt
```

입력된 파일명 "score.txt"로부터 성적 데이터를 읽고 각 학생의 평균 점수 및 과목 평균, 과목최고점을 받은 학생을 구한 후, 화면에 출력한다.

6.2. 아래 "입력파일 예제"처럼 성적 데이터 파일의 첫 번째 라인에는 전체학생수가 저장되어 있으며, 두 번째 라인부터는 각 학생의 정보가 아이디, 중간고사, 기말고사 점수 순으로 구성되어 있다. 각 데이터는 띄어쓰기(' '), 탭('₩t') 도는 줄 바꿈 ('₩n')으로 구분된다.

<입력파일 예제>

```
4
julie   89  97
harry   70  90
amanda  89  43
sarah   65  42
```

<출력화면 예제>

```
julie   93.80
harry   82.00
amanda  61.40
sarah   51.20
==============================
과목평균  : 72.10
과목최고점: 93.80 (julie)
```

6.3. 각 학생의 성적은 struct 를 이용해서 처리한다.

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

typedef struct
{
    char stu_id[20];
    int mid;
    int final;
    float ave;
}STUDENT;

STUDENT get_stu(FILE* input);
void print_list(STUDENT* list, int num);
float calc_ave(STUDENT* list, int num);
STUDENT* find_max_ave(STUDENT* list, int num);

int main(int argc, char *argv[])
{
    FILE *infile;
    STUDENT *stu_list, *max_stu;
    float tot_ave;
    int num, i;

    infile = fopen( ___argv[1]___, __"r"__ );
    if(infile == NULL)
    {
        printf( "Could not open input file\n" );
        return 100;
    } // if open fail
```

```
        // 파일의 첫 번째 라인에서 학생 수 읽기

    fscanf(__infile__, "%d", &num);

    //학생의 정보를 저장하기 위해 메모리 동적 할당

    stu_list = ____(STUDENT*)malloc(num*sizeof(STUDENT))_______;

    for(i = 0; i < num; i++)

        ___stu_list[i]__ = get_stu(infile);

    fclose(infile);

    //학생(아이디와 평균) 리스트 전체 출력 - 함수 print_list 호출

    ____print_list(stu_list,num)____;

    //과목 평균 최고점수 학생 찾기 – 함수 find_max_ave 호출

    ____max_stu=find_max_ave(stu_list,num)_______;

    //전체 평균 계산 – 함수 cal_ave 호출

    _____ave=cal_ave(stu_liet,num)________;

    printf("============================\n");
    printf("과목평균  : %.2f\n", tot_ave);
    printf("과목최고점: %.2f (%s)\n", ___max_stu->ave,max_stu->stu_id___);

    //동적할당 해제

    ___free(stu_list)___;

    return 0;
} //main

//입력 파일로부터 데이터 한 줄(아이디, 중간점수, 기말점수) 읽어서 s 에 저장
//각 학생의 평균은 중간고사 40%, 기말고사 60% 비율로 계산하여 저장
STUDENT get_stu(FILE* input)
{
    STUDENT s;

    fscanf(_input__, "%s %d %d", _s.stu_id,&s.mid,&s,final_);

    __s.ave__ = __s.mid__ *0.4 + _s.final__ *0.6;

    return s;
} //get_stu
```

```
void print_list(STUDENT* list, int num)
{
    int i;

    for(i = 0; i < num; i++)

        printf( "%s\t%.2f\n", _list[i]->stu_id,list[i]->ave__);
} // print_list

float calc_ave(STUDENT* list, int num)
{
    int i;
    float ave=0;

    for(i = 0; i < num ; i++)

        ____ave+=list[i]->ave_______;

    return ave / num;
} // calc_ave

//학생 목록에서 평균점수가 가장 높은 학생을 찾아서 그 학생의 주소를 반환한다.
STUDENT* find_max_ave(STUDENT* list, int num)
{
    int i;
    STUDENT *pmax = __&list[0]__;//list 도 가능

    for(i = 1; i  < num ; i++)
    {
        if(____pmax->ave____<____list[i].ave____)

            ________pmax=&list[i]__________;
    }

    return pmax;
} // find
```

7. [8 점] 다음은 영화 정보를 출력하는 프로그램이다. 영화정보는 제목, 번호, 장르로 구성되며, structure array 를 사용하여 저장한다. 아래 실행예제와 같은 결과가 나오도록 프로그램을 완성하시오.

<실행예제>

```
Movie List:
1   x-man            SF
2   Edge of Tomorrow Action
3   Maleficent       Fantasy
4   Oculus           Horror
5   Gravity          SF
SF list:
1   X-men
5   Gravity
```

```
#include <stdio.h>

enum Genre {SF, Fantasy, Horror, Action};
typedef struct{
    char title[30];
    int movieID;
    enum Genre genre;
}Movie;

void printList(Movie* pList, int size);
void printSF(Movie* pList, int size);

int main(void)
{
    int numOfMovies;
    Movie mvList[] =
    {
        ______{“x-man”,1,SF}________,

        ______{“Edge of Tomorrow”,2,Action}________,

        ______{“Maleficent”,3,Fantasy}________,

        ______{“Oculus”,4,Horror}________,

        ______{“Gravity”,5,SF}________,
    };

    numOfMovies = sizeof(___mvList____)/sizeof(___Movie____);
    printList(mvList,numOfMovies);
    printSF(mvList,numOfMovies);
    return 0;
}

void printList(Movie* pList, int size)
{
    int i;
    char* GenreString[] = {“SF”, “Fantasy”, “Horror”, “Action”};
    printf(“Movie List: \n”);
    for(i=0; i<size; i++)
        printf(“%2d\t%-20s\t%s\n”, __pList[i].movieID___,__pList[i].title__,
        ______GenreString[pList[i].genre]______);
}

void printSF(Movie* pList, int size)
{
    int i;
    Movie *pMovie = pList;
    printf(“SF List:\n”);
    for(i=0; i<size; i++, pMovie++)
    {   // pMovie 를 이용하여 아래의 빈칸을 채우시오. (plist 사용할 수 없음)

        if(_____pMovie->genre==SF_____)

            printf(“%2d\t%-20s\n”, __pMovie->movieID__, __pMovie->title__);
    }
}
```

8. (5점) 아래 프로그램은 사용자로부터 문자열을 입력 받고, 입력 받은 문자열을 대문자로만 구성된 문자열 혹은 소문자로만 구성된 문자열로 반환한다. 프로그램의 나머지 부분을 작성하시오.

(ASCII 코드 - 소문자: 97(a) ~ 122(z), 대문자: 65(A) ~ 90(Z))

```
Input string: Hello World!!
Upper: HELLO WORLD!!
Lower: hello world!!
```

```
#include <stdio.h>

// str 문자열 내의 모든 문자를 대문자로 변환하는 함수
char* strToUpper(char *str)
{
    char *tmp = str;

    while( ___*tmp!='\0'____ )
    {
        if(____*tmp>96 && *tmp<123_____)

            _____*tmp-=32_____;
            tmp++;
    }

    return str;
}

// str 문자열 내의 모든 문자를 소문자로 변환하는 함수
char* strToLower(char *str)
{
    char *tmp = str;

    while( ____*tmp!='\0'____ )
    {
        if(_____*tmp<91 && *tmp>64_____)

            _____*tmp+=32_____;
            tmp++;
    }

    return str;
}

int main()
{
    char str[100];

    printf("Input string: ");
    gets(str);

    printf("Upper: %s\n",strToUpper(str));
    printf("Lower: %s\n",strToLower(str));

    return 0;
}
```

9. (15점) 다음은 스택을 배열이 아닌 **Linked List**로 구현하는 프로그램이다. 이 프로그램에서 스택은 정수형 변수 하나를 저장하며 양수만 저장한다고 가정한다. 또한, 스택은 **단방향 Linked List**로 구현이 되어 있으며 아래의 주석을 참고하여 각 함수의 빈칸을 채우고, 프로그램을 실행하였을 때 출력되는 결과를 적으시오. 참고로, 스택은 모든 삽입(push)과 삭제(pop)가 top(최신 항목 가리킴)에서만 일어난다.

[주의 사항]

- 스택 연산으로는 isEmpty, isFull, push, pop, top이 있으며, push, pop, top 함수에서 예외 처리를 할 때 함수 **isEmpty**와 **isFull**을 사용할 것.
- **MAX_STACK_SIZE**를 한번 이상 사용할 것.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_STACK_SIZE 3

typedef struct node
{
    int data;
    struct node* next;
}NODE;
typedef struct
{
    int count;
    NODE* top;
}STACK;

int isEmpty(STACK *s);
int isFull(STACK *s);
void push(STACK *s, int data);
int pop(STACK *s);
int top(STACK *s);
STACK* initStack();
void showStack(STACK *s);
void freeStack(STACK *s);

int main()
{
    STACK *s;

    __s=initStack()____; // 스택 생성 및 초기화
    showStack(s);
    push(s, 1);
    push(s, 2);
    push(s, 3);
    showStack(s);
    push(s, 4);

    pop(s);
    showStack(s);
    pop(s);
    pop(s);
    pop(s);
    push(s, 5);
    push(s, 6);
    showStack(s);
    freeStack(s);
    return 0;
}
```

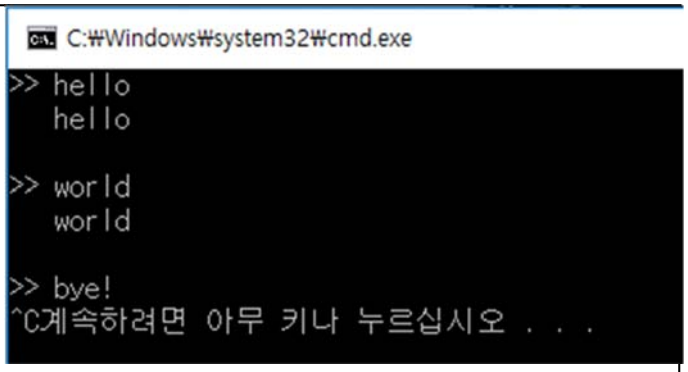
// 이 프로그램의 출력 결과를 쓰시오.

```
[Stack]:
[Stack]: 3 2 1
Overflow
[Stack]: 2 1
Underflow
[Stack]: 6 5
```

```
// 스택이 비어있는지를 확인한다. 스택이 빈 경우 1, 그렇지 않으면 0 을 반환한다.
int isEmpty(STACK *s)
{
    return ____s->count==0 ? 1:0_____;
}

//스택이 가득 차있는지를 확인한다. 스택이 가득 찬 경우 1, 그렇지 않으면 0 을 반환한다.
int isFull(STACK *s)
{
    return _____s->count==MAX_STACK_SIZE ? 1:0_______;
}

// 스택에 data(점수)를 추가한다. 스택이 가득 찬 경우 Overflow 를 출력한다.
void push(STACK *s, int data)
{
    NODE* pNew;

    if(___isFull(s)___) // 스택이 가득 찬 경우
        printf("Overflow\n");
    else
    {
        // 스택 노드 생성
        pNew = ____(NODE*)malloc(sizeof(NODE))____; // 동적 할당

        ___pNew->data_____ = data;

        ___pNew->next____ = _____s->top_____;

        ____s->top_____ = pNew;

        s->count++;
    }
} // push

// 스택에서 가장 마지막에 들어온 원소를 제거하고 그 원소 안에 저장된 값을 반환한다.
// 스택이 빈 경우 Underflow 를 출력하고 -1 을 반환한다.
int pop(STACK *s)
{
    NODE *pDel;
    int data = ____-1____;

    if(__isEmpty(s)___) // 스택이 빈 경우
        printf("Underflow\n");
    else
    {
        data = ___s->top->data___;

        pDel = ____s->top_____;

        _____s->top______ = _____s->top->next_____;

        s->count--;

        free(___pDel___);   //노드 삭제
    }

    return data;
} // pop
```

```
// 스택에서 가장 마지막에 들어온 원소 안에 저장된 원소를 반환한다.
// 스택이 빈 경우 Underflow를 출력하고 -1을 반환한다.
int top(STACK *s)
{
    int data = -1 ;

    if (___isEmpty(s)____) //
        printf("Underflow\n");
    else
        data = __s->top->data____;

    return data;
} // top

// 스택(헤드)을 위한 메모리를 동적 할당 받고 초기화 한다.
STACK* initStack()
{
    STACK *s;

    s = ___(STACK*)malloc(sizeof(STACK))_____;

    _____s->count____ = 0;

    ____s->top____ = NULL;

    return ___s____;

} // initStack

//스택의 내용을 최근에 추가된 원소의 데이터부터 순서대로 출력한다.
void showStack(STACK *s)
{
    NODE *temp;

    printf("[Stack]:");
    for(temp = __s->top___; __temp!=NULL___; ___temp=temp->next__)

        printf(" %d", ____temp->data____);
    printf("\n");
} // showStack

// 스택을 위해 동적할당 된 메모리를 할당해제 한다.
void freeStack(STACK *s)
{
    ...
}
```

10. (6 점) 큐(Queue)는 뒤(rear)에서는 새로운 항목을 삽입(enqueue)하고 앞(front)에서는 기존항목의 삭제(dequeue)가 일어나는 선형 리스트 이다. 단방향 링크드 리스트를 이용하여 큐를 구현하자.




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

// 큐 노드 구조체 NODE 선언
typedef struct node
{
    char data;
    struct node* next;
}NODE;

// 큐 헤드 구조체 QUEUE 선언
typedef struct
{
    int count;
    NODE *front;
    NODE *rear;
}QUEUE;

void printQueue (QUEUE *q);
int enqueue (QUEUE *q, char data);
int dequeue (QUEUE *q, char *data);

int main()
{
    int len, i, ioRes;
    char str[20];
    QUEUE queue; // 큐 (헤드)

    // 큐 (헤드) 초기화

    _queue.count=0_;

    _queue.front=NULL___;

    _queue.rear=NULL__;

    while(1)
    {
        printf(“>> ”);
        ioRes = scanf(“%s”, str);

        if(ioRes == EOF)
            break;

        len = strlen(str);
        for(i = 0; i < len; i++)

            enqueue( __&queue__, __str[i]__);
        printQueue(&queue);
    } // while
    printf(“bye!\n”);

    return 0;
} // main
```

```
// 큐가 빌 때까지 dequeue 함수를 호출한다.
void printQueue(QUEUE *q)
{
    char data;
    printf("  ");
    while( dequeue( _q__, _&data_ ) )
        printf("%c", data);
    printf("\n\n");
} // printQueue

// enqueue 성공시 1, 실패시 0 반환
int enqueue (QUEUE *q, char data)
{
    NODE *tmp;
    // 큐 노드 생성
    tmp = __(NODE*)malloc(sizeof(NODE))__;
    if(!tmp) // 동적 할당 실패시
        return 0;

    // 생성한 노드에 값 설정
    tmp->data = data;
    tmp->next = NULL;

    if(q->count == 0) //큐가 비어 있는 경우

        ___q->front=tmp_____;
    else
        ___q->rear->next=tmp_____;

    ___q->rear=tmp_________;
    q->count++;

    return 1;
} // enqueue

// dequeue 성공시 1, 실패시 0 반환
int dequeue (QUEUE *q, char *data)
{
    NODE *tmp;

    if (q->count ==0) // 큐가 비어 있는 경우
        return 0;

    *data = q->front->data;

    tmp = __q->front___;

    ____q->front___ = ____q->front->next____;
    (q->count)--;
    free( ___tmp___ );

    if(q->count == 0) // 큐에 1개 남은 노드를 삭제하는 경우
        q->rear = NULL;

    return 1;
} // dequeue
```

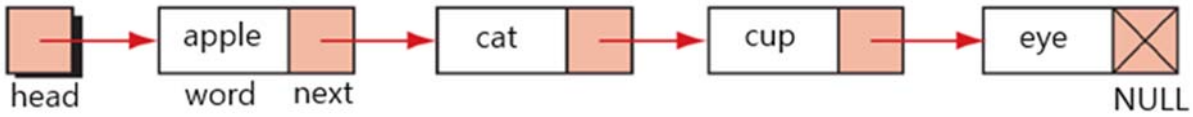
11. (5점) 회문(palindrome)은 'noon' 이나 'refer' 처럼 앞에서 읽으나 뒤에서 읽으나 동일한 단어나 구를 말한다. 아래의 프로그램은 입력한 문장이 회문이면 그 문장을 출력하고 회문이 아니면 회문을 만들어서 출력한다. 아래의 출력 결과가 나오도록 빈 칸을 채우시오.

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

//입력받은 문자열이 회문 인지 아닌지를 판단하는 함수로 회문이면 1을, 아니면 0을 반환한다.
int isPalindrome(char *str)
{
    int n, length = strlen (str);

    for ( n = 0; n < _length/2_; n++)
    {
        if ( __str[n]!=str[length-n-1]__ )
            return 0;
    }
    //length를 2로 나누지 않고 끝까지 검사해도 무방함
    return 1;
}

//입력받은 문자열을 회문으로 만들어서 반환한다.
char* makePalindrome(char* str)
{
    int n, length = strlen(str);

    for ( n = 0; n < _length-1_; n++)
    {
        __str[2*length-2-n]=str[n]__;

    ___str[2*length-1]='\0'____;
    return str;
}

int main(void)
{
    char inStr[][255] = { "noon", "refer", "mad", "race" };
    int i;
    for(i=0; i<4; i++)

        if ( __isPalindrome(inStr[i])___ )

            printf("%s\n", ___inStr[i]____);
        else
            printf("%s\n", ___makePalindrome(inStr[i])____);
    }
    return 0;
}
```

출력결과

```
noon
refer
madam
racecar
```

12. (17점) 다음은 단방향 선형 연결 리스트에 영어 단어를 알파벳 순으로 정렬되도록 삽입/삭제하는 프로그램이다. 아래의 그림과 주석을 참고하여 프로그램을 완성하시오.




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

typedef struct node
{
    char word[20];
    struct node* next;
}NODE;

//새로운 노드를 생성하여, newWord와 pNext로 값을 설정하고 생성한 노드를 반환한다.
NODE* newNode(NODE* pNext, char* newWord)
{
    NODE* pNew = _(NODE*)malloc(sizeof(NODE))___;

    ___strcpy(pNew->word,newWord)___;

    ___pNew->next____ = pNext;
    return pNew;
}

//새로운 단어를 알파벳 순을 유지하여 리스트에 삽입한다.
NODE* addNode(NODE* head, char* newWord)
{
    NODE *pos;

    //리스트가 비어있거나 리스트의 처음에 삽입하는 경우

    if(head==NULL || ___strcmp(head->word,newWord)==1___)

        ___head___ = newNode ( __head__, newWord);

    else //리스트의 중간이나 끝에 삽입하는 경우
    {
        for(pos = _head_; _pos->next!=NULL_; pos = pos->next)
        {
            if(__strcmp(pos->word,newWord)==1___)
                break;
        }
        _pos->next_ = newNode ( _pos->next_, newWord);
    }

    return head;
}
```

```
// wordToDelete와 일치하는 word를 가진 노드 삭제
NODE* deleteNode(NODE* head, char* wordToDelete)
{
    NODE *pDel, *pPre;

    //리스트의 첫번째 노드를 삭제하는 경우
    //즉, 첫번째 노드의 word와 wordToDelete가 일치
    if( __strcmp(head->word,wordToDelete)==0____)
    {
        pDel = __head___;

        ___head___ = ____head->next___;

        free( __pDel__ );
    }
    else //리스트의 중간이거나 마지막인 노드를 삭제하는 경우
    {
        for(pPre=head; __pPre->next!=NULL___; pPre = pPre->next)
        {
            if(___strcmp(pPre->next->word,wordToDelete)==0____)
            {
                pDel = _pPre->next___;

                __pPre->next___ = ___pDel->next____;

                free( __pDel__ );
                break;
            }
        }//for
    }

    return head;
}

// 리스트에서 맨 앞에 위치한 노드부터 메모리 할당을 해제한다.
// 리스트에서 맨 앞에 위치한 노드부터 삭제한다.
void deleteList(NODE *head)
{
    while(head)
        head = deleteNode(__head___, __head->word__);
}

// 리스트의 처음부터 끝까지 word를 순서대로 출력한다.
void printList(NODE* head)
{
    ...
}
```

```
int main(void)
{
    NODE *head=NULL;
    char command[7];
    char word[20];

    while(1)
    {
        printf("Command: ");
        scanf("%s", command);

        // add 명령어를 입력한 경우
        if( __strcmp("add",command)==0___)
        {
            scanf("%s", word);

            _head=addNode(head,word)_____; // addNode 함수 호출
        }
        // delete 명령어를 입력한 경우
        else if( ___ strcmp("delete",command)==0_____)
        {
            scanf("%s", word);

            ___head=deleteNode(head,word)__; // deleteNode 함수 호출
        }
        // quit 명령어를 입력한 경우
        else if( ____ strcmp("quit",command)==0_____)
            break;

        printList(head);
    } // while
    deleteList(head);

    return 0;
}
```

실행예제) (밑줄은 사용자 입력을 말함)

```
Command: add cat
- (cat)
Command: add apple
- (apple) - (cat)
Command: add eye
- (apple) - (cat) - (eye)
Command: add cup
- (apple) - (cat) - (cup) - (eye)
Command: delete apple
- (cat) - (cup) - (eye)
Command: quit
계속하려면 아무 키나 누르십시오 . . .
```