C 프로그래밍 중급

Project

❑ 제출 방법

"영문이름_사번" 으로 폴더를 만들고, 폴더안에

"project1.c, project2.c….." 로 형태로 소스를 넣어 주세요. 소스만 포함해주시면 됩니다. 실행파일은 없어도 됩니다.

readme**.txt** 을 만들고 몇 번 과제를 해결했는지 적어 주세요. (예시 : 1~3번까지 완성)

과제는 일부만 해결해도 부분 점수가 부여 됩니다.

폴더를 포함해서 통째로 압축해서 LGE MOOC "Project#1"에서 "select file" 버튼을 눌러서 압축파일을 올려 주세요

주의 사항 : 압축된 파일명은 반드시 영문이어야 합니다.

❑ 제출 기한

과정 마감일 까지 (강좌 계획 참고)

과제 1. map 구현하기

아래 코드가 동작할 수 있도록 map 함수를 만들어 보세요.

```c
#include <stdio.h>

int foo(int a)
{
    return a * 2;
}

int main(void)
{
    int x[10] = { 1,2,3,4,5,6,7,8,9,10 };
    int y[10] = { 0 };

    // x ~ x+10(x+10 포함 안됨)을 foo로 전달해서 foo의 반환값을 y에 넣습니다.
    map(foo, x, x + 10, y); // 2,3 번째 인자는 배열의시작과 마지막 다음요소의 주소입니다

    for (int i = 0; i < 10; i++)
    {
        printf("%d\n", y[i]); // 2, 4, 6, 8, 10, 12, 14, 16, 18, 20
    }
}
```

과제 2. execute_function 만들기

아래 main 함수가 사용하는 execute_function 함수를 구현해 보세요.

```
int foo(int a) { return 2 * a;}
int goo(int a) { return a + 1;}
int hoo(int a) { return a * 5; }


int main(void)
{
    int n = execute_function(1, &foo, &goo, &hoo, 0); // hoo(goo(foo(1))) 처럼
                    // 동작해야 합니다.단, 함수의 갯수는 변경될수 있어야 합니다.
                    // 0은 더이상 함수가 없다는 의미.

    printf("%d\n", n); // 15

    int n2 = execute_function(3, &foo, &goo, 0); // goo(foo(3))
}
```

과제 3. 배열 결합

2*2의 2차원 배열 2개를 인자로 받아서 2*2*2 의 3차원 배열을 만들어서 반환하는 “make_3d_arr_from_2d_arr” 함수를 만들어 보세요.

3차원 배열은 힙에 만들어서 반환해 주세요.

Typedef를 사용하지 말고 타입을 직접 표기해서 작성해 보세요.

아래 main 함수가 실행될 수 있도록 “make_3d_arr_from_2d_arr” 제공하면 됩니다.

```
int main(void)
{
    int x[2][2] = { 0 };
    int y[2][2] = { 0 };

    // 반환된 배열을 동적메모리 할당을 사용해 주세요.
    int(*z)[2][2] = make_3d_arr_from_2d_arr(x, y);

    // z의 모든 내용을 출력해 주세요.

    free(z);
}
```

과제 4. Generic linked list 활용

아래 코드는 간단한 리스트 코드입니다.

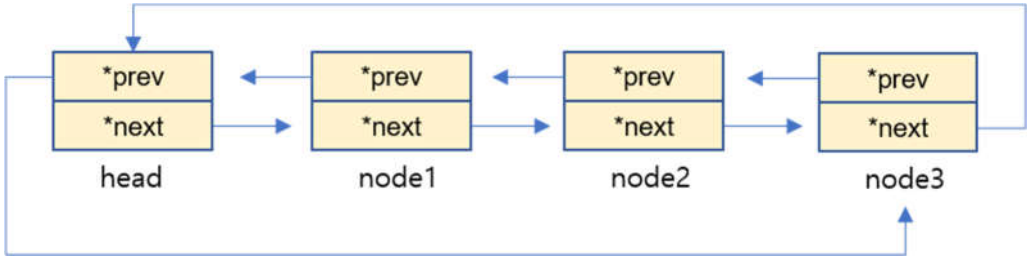
List 자체 코드를 완전히 이해 하지 못해도 과제는 해결할 수 있습니다.

아래 코드에서 NODE 구조체는 저장하는 데이터는 없고 2개의 포인터(next, prev)만 저장하고 있습니다.

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

struct _Node
{
    struct _Node* prev;
    struct _Node* next;
};
typedef struct _Node NODE;

// 리스트의 헤드를 초기화 하는 함수
void init_list(NODE** head)
{
    (*head)->next = *head;
    (*head)->prev = *head;
}
// 2개의 노드 사이에 새로운 노드를 넣는 코드 입니다.
void insert_node(NODE* node, NODE* prev, NODE* next)
{
    node->prev = prev;
    node->next = next;
    prev->next = node;
    next->prev = node;
}
void insert_front(NODE** head, NODE* node)
{
    node->next = 0;
    node->prev = 0;
    insert_node(node, *head, (*head)->next);
}
void insert_back(NODE** head, NODE* node)
{
    node->next = 0;
    node->prev = 0;
    insert_node(node, (*head)->prev, *head);
}
int main()
{
    NODE* head =  (NODE*)malloc(sizeof(NODE));
    NODE* node1 = (NODE*)malloc(sizeof(NODE));
```

```
    NODE* node2 = (NODE*)malloc(sizeof(NODE));
    NODE* node3 = (NODE*)malloc(sizeof(NODE));

    init_list(&head);
    insert_front(&head, node1);
    insert_front(&head, node2);
    insert_front(&head, node3);

    free(node1);
    free(node2);
    free(node3);
    free(head);
}
```

위 코드에서 main 함수가 실행되었을 때의 메모리 그림은 다음과 같습니다.



위 코드의 핵심중 하나는 list의 NODE가 실제 저장하는 데이터는 없고 단지, prev, next 만 있다는 점입니다.

아래 코드는 위 list 를 활용하는 예제 입니다.

아래 코드가 실행 될 수 있도록 make_people, print_vip, print_reserve 함수를 만들어 보세요.

온라인 강좌에서 배웠던 내용중의 한가지를 활용하게 되는 과제 입니다.

```
typedef struct _People
{
    char name[16];
    int  age;
    NODE vipNode;
    char phone[20];
    NODE reserveNode;
} PEOPLE;

int main()
{
    NODE* viphead = (NODE*)malloc(sizeof(NODE));
    NODE* reservehead = (NODE*)malloc(sizeof(NODE));
```

```
    init_list(&viphead);
    init_list(&reservehead);

    PEOPLE* p1 = make_people("kim1", 30, "010-1234-5678");
    PEOPLE* p2 = make_people("kim2", 30, "010-1234-5678");
    PEOPLE* p3 = make_people("kim3", 30, "010-1234-5678");
    PEOPLE* p4 = make_people("kim4", 30, "010-1234-5678");

    insert_front(&viphead, &p1->vipNode);
    insert_front(&viphead, &p3->vipNode);
    insert_front(&viphead, &p4->vipNode);
    insert_front(&viphead, &p2->vipNode);

    insert_front(&reservehead, &p2->reserveNode);
    insert_front(&reservehead, &p4->reserveNode);
    insert_front(&reservehead, &p1->reserveNode);
    insert_front(&reservehead, &p3->reserveNode);

    print_vip(viphead);         // "kim2", "kim4", "kim3", "kim1"
    print_reserve(reservehead); // "kim3", "kim1", "kim4", "kim2"
}
```

과제 5. 통보 센터 만들기

구조체 안에 함수 포인터를 멤버로 사용하면 C 언어에서도 C++ 언어와 같은 멤버 함수 개념을 만들 수 있습니다. 아래 코드에서 main 함수의 동작 방식을 잘 생각해 보세요.

```
#include <stdio.h>

typedef struct _Point
{
    int x;
    int y;
    void(*set)( struct _Point*, int, int);
    void(*destroy)( struct _Point*);
} Point;
void set(Point* this, int a, int b)
{
    this->x = a;
    this->y = b;
}
void destroy(Point* this)
{
    free(this);
}
Point *make_point()
{
    Point* p = (Point*)malloc(sizeof(Point));   // 1. 메모리 할당
    p->set      = &set;                         // 2. 멤버 함수 연결
    p->destroy  = &destroy;
    return p;
}
int main()
{
    Point *p1 = make_point();
    p1->set(p1, 10, 20);        // p1->x = 10, p1->y = 20 의 의미 입니다.
    p1->destroy(p1);
}
```

위 코드를 참고 해서 아래의 main 함수가 실행될 수 있도록 통보 센터를 만들어 보세요.

```
void foo(int value) {}
void goo(int value) {}
void hoo(int value) {}

int main()
{
```

```
    // 통보 센터는 키 값(문자열)을 가지고 함수를 등록합니다
    // 특정 상황이 발생하면 등록된 함수가 호출되어야 합니다.
    NotificationCenter* nc = make_notification_center();


    // 통보 센터에 키값과 함수를 등록합니다
    // 키값은 최대 10개 까지 등록 가능 해야 하고
    // 하나의 키값에는 최대 10개의 함수가 등록가능해야 합니다.
    // 자료구조는 배열을 사용하면 됩니다
    nc->addObserver(nc, "LOWBATTERY", &foo);    // "LOWBATTERY" 키값으로 &foo 등록
    nc->addObserver(nc, "LOWBATTERY", &goo);    // 동일한 키값으로 2번째 함수 등록
    nc->addObserver(nc, "DISCONNECT", &hoo);    // 새로운 키값으로 &hoo 함수 등록

    nc->PostNotification(nc, "LOWBATTERY", 30); // LOWBATTERY 키값으로 등록된
                                        // 모든 함수를 호출합니다.
                                        // 등록된 함수에 전달할 인자는 30 입니다.

    // 통보 센터를 파괴 합니다.
    nc->Destroy(nc);
}
```