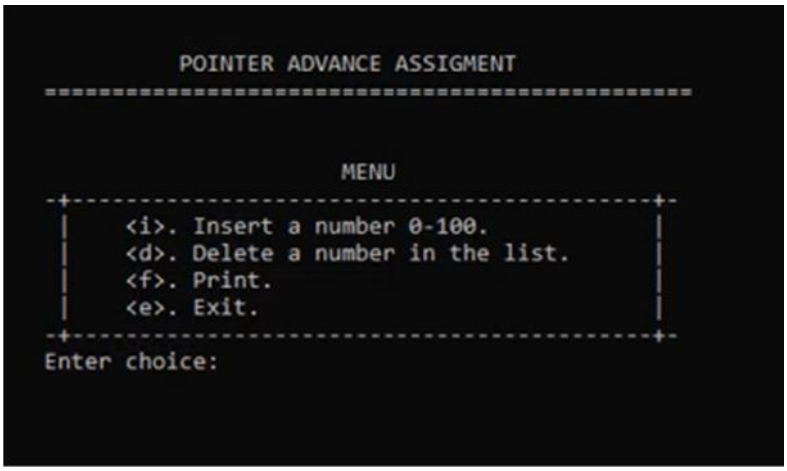
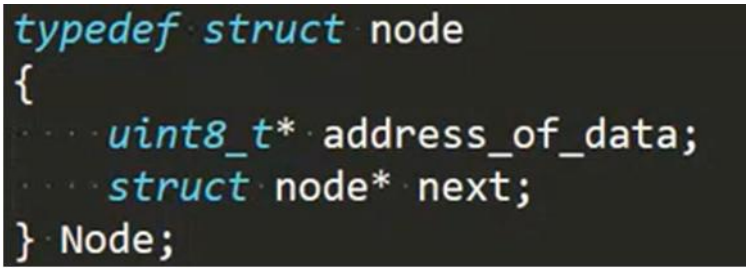
Ko nên include bất cứ thứ gì trong file.h mà phải include trong file.c vì mai sau sẽ có việc là chống include chéo.

Trong bài pointer advanced khi mà viết code các thứ thì phải chia tầng chia thư viện, chia làm phần application các thứ thì phần gọi sẽ gọi các application, application sẽ gọi trực tiếp các phần dưới và từ thư viện sẽ gọi qua lên trên thông qua call back và khi include trong file.h thì nó có thể 2 thư viện include lẫn nhau dẫn đến file.c của thằng này include file.c của thằng khác, 2 bên sẽ include chéo nhau dẫn đến ko nên chứ ko phải ko được. trong 1 số tình huống bắt buộc phải include trong file.h nhưng mà ko nên

Khi biên dịch 1 file.c một thì nó sẽ biên dịch tuần tự từ trên xuống dưới ví dụ với trước khi biên dịch file linkedlist.h thì nó sẽ biên dịch 2 file .h trước. Khi biên dịch 2 file.c này sau đó include cái stdint.h và stdlib.h trước cái file.h trước thì sau đó nó sẽ biên dịch file.h





Việc đặt tên: [tên module][tên của struct][struct để phân biệt với enum]_t -> là 1 kiểu typedef]

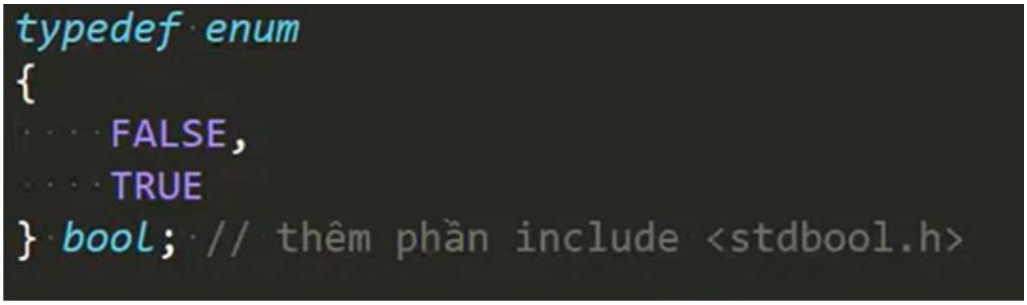
Thay vì đặt tên như hình trên thì đặt tên ntn

Linkedlist_node_struct_t;

Khi nhìn vào tên người ta sẽ có được thông tin là đây là 1 kiểu struct, là 1 kiểu được người lập trình định nghĩa, cái chữ t chính là viết tắt của từ typedef, struct để hiểu đây là 1 kiểu struct, chữ node là tên của 1 struct đấy. khi đọc tên thì người ta biết struct này

dành cho danh sách liên kết này. Viết cái node ko như hình trên thì ko lẽ đặt tên tất cả đều là node? Node đó thuộc module nào?

Cho nên là đầu tiên đặt tên module bằng tên chính file đó



Linkedlist_bool_t;

Kiểu typdef thì có _t

Để người khác đọc code cho dễ hiểu.

Ko chia sẻ biến toàn cục cho nhiều modules khác nhau để chuyên biệt hoá chức năng của module tốt hơn.

Khi mà chia sẻ các biến toàn cục dẫn đến các modules sử dụng chung 1 biến toàn cục dẫn đến modules ko tốt vì chúng dính dáng đến nhau qua biến toàn cục kia dẫn đến 1 yêu cầu là ko được chia sẻ bất kỳ biến toàn cục nào giữa các modules nên việc extern là ko được cho phép.

Thay vì vậy thì được phép dùng biến toàn cục trong 1 file -> biến static

Insert_After: insert 1 node vào đằng sau 1 node khác thì node kia đã có con trỏ next đến node tiếp theo thì chỉ việc cho node mới trỏ đến node mà cái node kia đang trỏ đến và node kia lại trỏ vào node mới.

Ko dùng int vì ko tối ưu

Hàm cung cấp phải đúng yêu cầu đề bài, nếu 1 hàm ko xử lý yêu cầu đề bài mà ở đây ví dụ là phục vụ cho việc xử lý linked list -> hàm này ko public mà nó phải static.

Cung cấp API cho người dùng thì nó phải đi theo các yêu cầu của người dùng

Người dùng yêu cầu nhập, phải có hàm nhập, xoá, print, mà những hàm kia chỉ để xử lý nội bộ các hàm bên trong thư viện, thao tác nội bộ trong thư viện là phải để static.

Ví dụ ở đây là các hàm dùng để phục vụ trong nội bộ của thư viện để xử lý các thao tác với danh sách liên kết, mà không phục vụ yêu cầu người dùng, tất cả phải là static, cung cấp hàm nào là public, hàm nào là static.

Đề bài là người dùng muốn nhập 1 giá trị vào vị trí nào đấy trong 1 mảng, vậy việc đầu tiên là có 1 hàm để nhập enter hoặc insert bất kỳ cái gì đó, có 1 hàm có giá trị trả về mã lỗi gì đấy, tên hàm là gì,

<mã lỗi> nhập (vị trí, giá trị) cần có thám số vị trí và giá trị

{

Lỗi nếu vị trí không đúng;

Vị trí đã được nhập;

Giá trị không trong dải;

Giá trị đã tồn tại;

Thành công;

}

Cho phép người dùng nhập vào vị trí nào đấy của mảng có giá trị vị trí ở đây từ 0 đến 19. Giá trị từ 0 đến 100. Và trong đề bài sẽ mô tả nếu báo lỗi là vị trí không đúng, vị trí đã được nhập, giá trị không trong dải, giá trị đã tồn tại thì hàm nhập mới có mã lỗi như thế kia. Sau đấy thì đề bài yêu cầu cho phép xoá

Thì có hàm xoá:

<mã lỗi> xoá (giá trị)

{

Lỗi nếu giá trị không tồn tại trong mảng;

Hoàn thành

}

Tiếp theo cung cấp 1 hàm print theo 2 trường hợp, 1 là in tất cả các giá trị, 2 là in giá trị đã được sắp xếp.

<mã lỗi> print (option)

Tham số option để print toàn bộ hoặc print mảng đã sắp xếp

{

3: option không đúng;

Thành công.

}

Những hàm vừa viết kia là những hàm public, còn việc xử lý danh sách liên kết thì người dùng ko cần quan tâm. Những hàm như thế này phải là static.

Nếu hàm này chỉ được gọi duy nhất 1 lần thì ko nên là 1 hàm

Viết 1 hàm insert nếu viết tốt thì chỉ cần 10 – 15 dòng code.

Tất cả các biến phải khai báo từ khi bắt đầu hàm

// phần khai báo biến

// code

//return ở đây

Khai báo biến ở trong hàm thì nó sẽ vào stack, từ đó việc quản lý stack sẽ dễ dàng hơn. Hàm này chiếm 5 biến thì stack chiếm bao nhiêu, thay vì nhìn vào code xem thế nào.

Sau này phát triển 1 tool đo stack thi chỉ cần đếm biến ở đầu hàm, thân hàm ko quan tâm

Người dùng quan tâm vị trí, giá trị, ko quan tâm đến node

Những cái người dùng ko quan tâm thì ko xuất hiện trước người dùng (trong các file ứng dụng)

```
đề bài là tạo 1 menu -> ứng dụng là xây dựng 1 menu.
- 4 option
    1: Cho phép người dùng nhập.
        Đề nghị nhập vị trí.
            5.
        Đề nghị người dùng nhập giá trị.
            50
        -> call đến hàm của thư viện
        mã lỗi = nhập (5, 10);
        if (thành công = mã lỗi)
            in người dùng nhập giá trị 10 vào ô thứ 5 thành công.
        else (vị trí lỗi = mã lỗi)
            in vị trí người dùng nhập không đúng.
        else (giá trị tồn tại ... = mã lôi)
            in giá trị đã tồn tại
    2: Cho phép người dùng nhập
        Đề nghị người dùng nhập giá trị để xóa.
            10
        -> call đến function để xóa.
        mã lỗi = xóa (10);
        if (mã lỗi = giá trị không tồn tại trong mảng)
            in thông tin: giá trị muốn xóa không tồn tại trong mảng.
```

```
3: In thông.
    Đề nghị người dùng nhập option muốn in, 0 = in toàn bộ, 1 = in các giá trị
        0
    -> call đến hàm inthoong
    mã lỗi = print(0);
    if mã lỗi/
        print thông tin lỗi.
4: Kết thúc chương trình.
```

Vậy ứng dụng có 1 hàm menu -> menu.c

Có 1 hàm menu (void)

Void menu (void)

{

Các option

}

Main.c

Call menu và chạy

Trong hàm menu có các hàm nhập giá trị từ bàn phím

Static Uint8_t (nhập giá trị từ bàn phím (void)

{

uint8_t retVal

Uint32_t giá trị nhập

Scan(…& giá trị nhập);

If giá trị nhập lớn hơn 0 và bé hơn 256 thì return retVal = giá trị nhập không thì báo lỗi

Chia làm main.c, menu.c.h, 2 file lib.c và .h cung cấp các hàm public

Các hàm xử lý danh sách liên kết sẽ làm static , static vì hàm main ko quan tâm tới việc nhập từ bàn phím, hàm này chỉ sử dụng trong menu thì là static, khai báo nội bộ trong thư viện

Optimize tối đa

```
typedef struct Node
{
    uint8_t data;   // thua, lang phi bo nho
    struct Node* next;              /* Note: Naming Rule: [ModuleName][StructName][Struct or Enum][_t] */
} *NodePtr, Node;                   /* [_t] means that it is a typedef.                                */
                                    /* Example: API_Node_Struct_t                                      */
```

Có mảng để lưu trữ data rồi nên ko cần thêm data ở node làm gì

Tên hàm cũng phải có tên module trước, sau đến tên chức năng của hàm. Tên module thường trùng tên file. Đặt file là API thì ở hàm đặt Api_insert

Thì khi nhìn vào hàm thì người ta nhìn ra là hàm này thuộc module API và thuộc chức năng insert

[ModuleName]_[FunctionName]

Các hàm static ko thuộc chức năng của public

```
void show_menu(void);
```

Thuộc ứng dụng ko thuộc thư viện

```c
#ifndef _API_H_
#define _API_H_

/*****************************************************************************
* Definitions
*****************************************************************************/
typedef struct Node
{
    uint8_t data;   // thua, lang phi bo nho
    struct Node* next;              /* Note: Naming Rule: [ModuleName][StructName][Struct or Enum][_t] */
} *NodePtr, Node;                   /* [_t] means that it is a typedef.                                */
                                    /* Example: API_Node_Struct_t                                      */
/*****************************************************************************
* API
*****************************************************************************/
/**
*   @brief Insert a new node to the list in ascending order of value
*   @param head - Address of the pointer that is the head pointer of the list
*          new_node - Pointer to the new node want to insert.
*   @return true if insert SUCCESSED, false if insert FAILED.
*/
bool insert(NodePtr* head, NodePtr new_node);  // [ModuleName]_[FunctionName]
bool del(NodePtr list, uint8_t data);
void print_arr(uint8_t* arr); // arr hay list phai la 1 bien trong thu vien
void print_list(NodePtr list);
```

Style thì nên như thế này là được

Cần mô tả trong tất cả các hàm, ko chỉ 1 hàm

Thuật toán tối ưu ko cần check

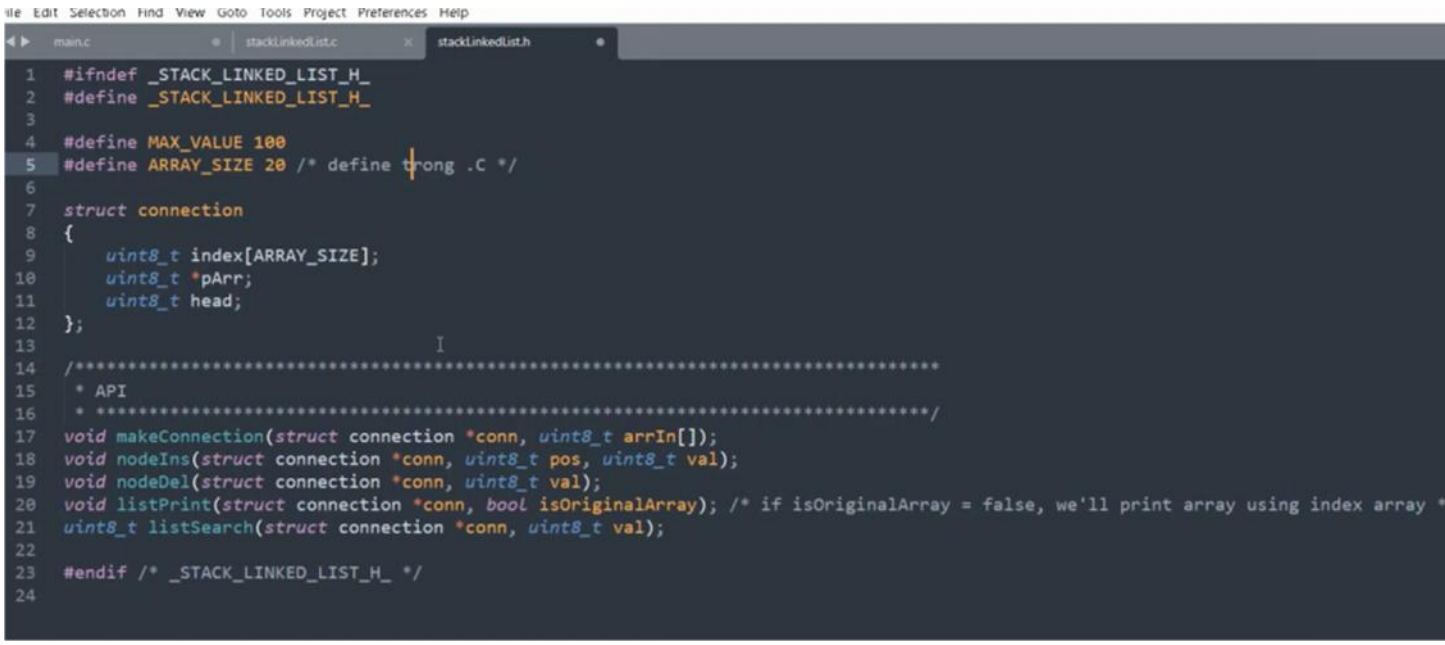
Cover dc cả NULL hay ko NULL

Else if else ko cần luôn

```c
printf("\t                    MENU\n");
printf("\t-+-----------------------------------------+-\n");
printf("\t |    <1>. Insert a number 0-100.          |\n");
printf("\t |    <2>. Delete a number in the list.    |\n");
printf("\t |    <3>. Print.                          |\n");
printf("\t |    <4>. Exit.                           |\n");
printf("\t-+-----------------------------------------+-\n");
printf("\tEnter choice: ");
```

```c
#define MAX_VALUE 100
#define ARRAY_SIZE 20
```

Người dùng ko cần quan tâm 2 cái này mà cho nên define nó trong file.c



Typedef struct thì nên define trong file.c