LAB 13

> 本次LAB目标：
>
> 1. 介绍指针、结构体相关知识
> 2. 了解变量作用域
> 3. 布置课堂练习

获取及提交Lab

获取：通过 `https://github.com/fdu-20ss-programming-A/lab13` 或者 超星平台 获取。

指针、结构体

1. 指针与指针形式的高维数组

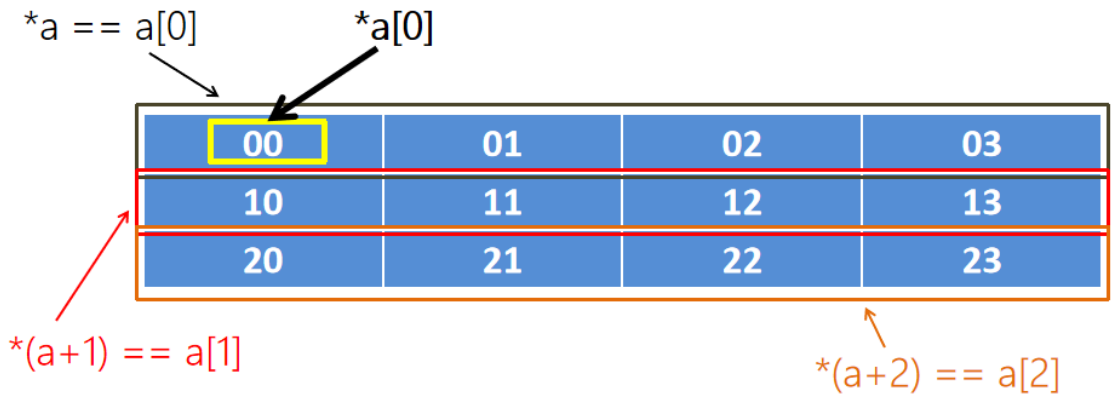
float *p——定义的是：一个指向浮点数的指针，在使用过程中，`p`指向的是一个浮点数字

int a[3] [4] = {{65,67,70,60},{80,87,90,81},{90,99,100,98}};——定义的是：一个3行、4列的二维数组，数组的元素为int。

在调用的过程中，`a`就是这个数组的首地址，因此`*a`指向的是数组`{65,67,70,60}`。`a+1`指向的是数组`{80,87,90,81}`

`*a`等于`a[0]`，`*(a+1)`等于`a[1]`，`*(a+2)`等于`a[2]`

`a[0], a[1], a[2]`又都是各个一维数组的首地址，比如`a[0]`就等同于`&a[0][0]`，因此`*a[0] = a[0][0]`



int (*p)[4]——括号中的`*`表明 p 是一个指针，它指向一个数组，数组的类型为`int [4]`。因此，如果令`int (*p)[4] = a`，则此时p指向a[0]，p+1指向a[1]，p+2指向a[2]

int *p[4]——`[]`的优先级高于`*`，因此该形式应理解为`int *(p[4])`，括号里面说明`p`是一个数组，包含了4个元素，括号外面说明每个元素的类型为`int *`

> 建议阅读C语言二维数组指针与C语言指针数组

2. 结构的声明、赋值

- 声明结构体

```
结构体的一般形式为：

struct  结构体名
{
    数据类型       成员名1；
    数据类型       成员名2；
     :
    数据类型       成员名n；
 }；
```

- 定义结构体变量

  先定义结构体类型**再**定义变量名，这是C语言中定义结构体类型变量最常见的方式。

```
struct 结构体名
{
     成员列表；
}；

struct 结构体名 变量名；
```

  在定义类型的**同时**定义变量。这种形式的定义的一般形式为：

```
struct 结构体名
{
     成员列表；
}变量名；
```

  直接定义结构类型变量，其一般形式为：

```
struct       //没有结构体名
{
    成员列表；
}变量名；
```

- 结构体的赋值
  - 结构体的初始化

    结构体变量在初始化时应该用花括号“{}”将用来初始化的各个数值括起来，并且各个数值之间要用逗号分隔开来，同时必须保证数值与相应成员变量的类型要一一对应。

```
#include <stdio.h>
struct Init{
    int a;
    double b;
    char *c;
    float d;
}id1 = {1,2.0,"hello"},id2[2],*id;
```

- 结构体的普通赋值
  - 有序赋值

```
#include <stdio.h>
struct Init{
    int a;
    double b;
    char *c;
    float d;
}id2;

struct Init id2 = {1,1.1,"hello"};  //这种写法默认float为空
```

  - 无序赋值（键值对）

    利用成员运算符进行赋值。可分为结构体内部赋值和结构体外部赋值.
    这种赋值方式涉及结构体成员的引用。
    引用方式为：结构体变量名.成员名。`.`是成员运算符，也称为分量运算符，它在所有运算符中优先级最高。

```
struct Init id2;
id2.a = 1;
id2.c = "world";
id2.d = 1.1;
```

```
struct Init id2 = {
    id2.a = -1,
    .c = "hello",
    .b = 4.0
};
```

- 补充
  - 如果成员本身又属于一个结构体变量类型，则要用若干个成员运算符，一级一级地找到最低一级的成员，并且只能对最低级的成员进行操作。
  - **可以将一个结构体变量直接赋值给另一个具有相同结构的结构体变量。**
  - 对成员变量可以像普通变量一样进行各种运算。
  - 可以引用成员的地址，也可以引用结构体变量的地址。结构体变量的地址主要用作函数参数，传递结构体的地址。

变量的作用域

```
#include <stdio.h>
int a = 10;//定义全局变量a，初值为10

void A(int i) {
    a *= i;//此处修改全局变量a
}
```

```
int B(int i) {
    static int a = 2;//定义静态变量a，只在B中有用，屏蔽全局变量a，只赋值一次
    a *= i;
    return a;
}

int main() {
    int a = 3; //定义本地变量a，只在main中有用，屏蔽全局变量a
    a = B(a);//调用B，修改静态变量为6，返回6，赋值给本地变量a，值为6
    A(a);//把6传给A，修改全局变量a为60，不影响本地变量
    printf("%d\n", a);//打印本地变量a，值为6
    return 0;
}
```

课堂练习

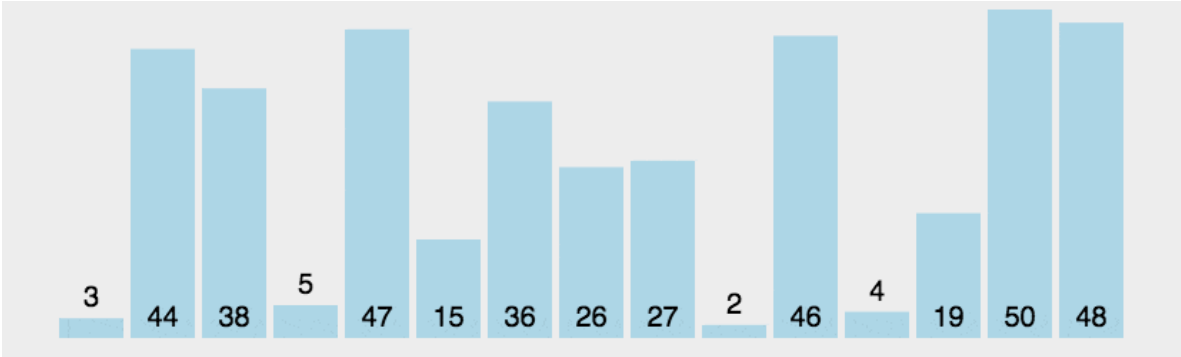
> 明天会发上一次作业和本次课堂练习的答案，课堂练习不需提交

Question 1 递归函数使用

> 输入一个字符串，打印出该字符串中字符的所有排列。
>
> 例如输入字符串abc，则输出由字符a、b、c所能排列出来的所有字符串abc、acb、bac、bca、cab和cba。
>
> 注意输入的字符串不包含重复的字符。
>
> 提示：使用递归，传递一个字符串，固定第一个字符，计算剩余字符的排列。调换其他字符与第一个字符的顺序，继续第一步。

Question 2 冒泡排序

> 冒泡排序（Bubble Sort）也是一种简单直观的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢"浮"到数列的顶端。



> 编写一个程序实现链表的冒泡排序算法（从小到大）

```
struct Node{
    int value;
    struct Node *next;
};

struct Node *bubbleSort(struct Node *head){
    return NULL;
}
```

例如：

> 输入链表为：
>
> 3->5->4->2->1->NULL
>
> 经过冒泡排序后，返回的链表为：
>
> 1->2->3->4->5->NULL

Question 3 单链表的顺序查找

> 编写一个程序，查找给定值在链表中的位置。
>
> 若给定值不在链表中，返回NULL，若在链表中，返回
>
> 链表内容和需要查找的元素均通过读取输入确定。
>
> 例如：
>
> 输入链表为：
>
> 7->9->2->1->4->5->NULL
>
> 要查找值为1的元素，则需依次比较7、9、2、1

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

typedef struct Node{
    int value;
    struct Node *next;
}Node, *Linklist;

Linklist InitList(Linklist list){
    list = (Linklist)malloc(sizeof(Node));
    list->next = NULL;
    return list;
}

Node *locateItem(Linklist list, int value){
    //TODO
}

int main(){
    int input, n;
    Linklist list;
    Node *r, *p;
    list = InitList(list);
```

```
    r = list;
    scanf("%d", &n);
    getchar();
    for (int i = 0; i < n; ++i) {
        scanf("%d", &input);
        getchar();
        p = (Linklist)malloc(sizeof(Node));
        p->value = input;
        p->next = NULL;
        r->next = p;
        r = r->next;
    }
    scanf("%d", &input);
    Node *result = locateItem(list, input);
    printf("%d", result->value);
    return 0;
}
```

Question 4 有序数组的二分查找

> 二分查找，也称为是折半查找，属于有序查找算法。
>
> 用给定值k先与中间结点的关键字比较，中间结点把线形表分成两个子表，若相等则查找成功；若不相等，再根据k与该中间结点关键字的比较结果确定下一步查找哪个子表，这样递归进行，直到查找到或查找结束发现表中没有这样的结点。
>
> 例如：
>
> 假设待查找数列为 1、3、5、7、9、11、19，我们要找的元素为 18，首先，我们找到中间的元素 7 （(0+6)/2=3)
>
> 
>
>
> 中间元素为 7，我们要找的元素比 7 大，于是在后半部分查找，现在后半部分数列为 9、11、19，我们找到中间元素，如下图所示
>
> 
>
>
> 中间元素为 11，与 11 比较，比 11 大，则继续在后半部分查找，后半部分只有一个元素 19 了，这时直接与 19 比较，若不相等，则说明在数列中没有找到元素，结束查找。
>
> 编写一个程序实现有序数组的二分查找

```
#include<stdio.h>
//基本思路：将按从小到大顺序排序好的数据存放到数组里
//         设置“前”“中”“后”标签，检查“中”指向的中间元素是否等于查找值
//         若查找值小于中间元素就将“后”变为原来的“中”，若大于就将“前”变为原来的“中”
//         继续比较“中”，比较，循环，直至找到查找值在数组中的位置，或循环结束。
int binsearch(int *sortedSeq, int seqLength, int keyData);

int main()
{
```

```c
    int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int location;
    int target = 4;
    location = binsearch(array, 9, target);
    printf("%d\n", location);
    return 0;
}

int binsearch(int *sortedSeq, int seqLength, int keyData)
{
    //TODO
}
```