前言

2020年8月15日 13:04

如何定义函数

- int add(int a, into);
- def add(a, b):
- 不会写咋办？ 从需求入手——先写调用

函数的返回值干啥用?

- 返回“我要的东西”（函数运行结果）
- 返回状态（函数成功与否）

递归函数

- 自己调自己？死循环、出不来？
- 别人写的看上去很简洁，可自己就是不会写
- 诀窍：信念
- 不要滥用递归（效率、栈溢出）

- 有一些长得像函数，其实不是函数（sizeof）

```c
#include <stdio.h>
int main(int argc, const char * argv[]) {

    int a = 666;

    printf("%ld\n", sizeof(int) );

    return 0;
}
```

放进去a或者int都有返回值4

```c
#include <stdio.h>
int main(int argc, const char * argv[]) {

    int a = 666;

    printf("%ld\n", sizeof(a = 2223) );
    printf("%d\n", a);

    return 0;
}
```

```
4
666
```

赋值语句没有执行，为何?
因为sizeof不是函数，如果是函数，则要对表达式求值然后传值
在这里，sizeof只是一个运算符
可以写成sizeof a ，小括号是可以不写的

2.1.1 数组的定义与使用

C语言

- int array[N] 或 int *array = malloc()
- “原生态”的数组，与“指针”关系密切
- 数组名 不是 指针 ！
- 如何传递参数?

数组名不是指针，sizeof(数组)不是指针大小
array[100]旁边是中括号，尊重括号，是一个数组
*(array+1)是表达式，表达式求值，array就是指针了
函数值传递，传递的是array的值（就是地址）

2.1.3 数组实例

高精度加法：

求两个100位的十进制数字之和

C：

我模拟竖式加法！

```c
int main(int argc, const char * argv[]) {

    int a[11] = {0,9,8,7,6,5,4,3,2,1};
    int b[11] = {1,2,3,4,5,6,7,8,9,9};
    int sum[11] = {0};
    int carry=0;
    for(int i=0; i<11; i++){
        int s;
        s = a[i] + b[i] + carry;
        carry = s/10;
        sum[i] = s%10;
    }
    for(int i=10; i>=0; i--)
        printf("%d", sum[i]);
    putchar('\n');

    return 0;
}
```

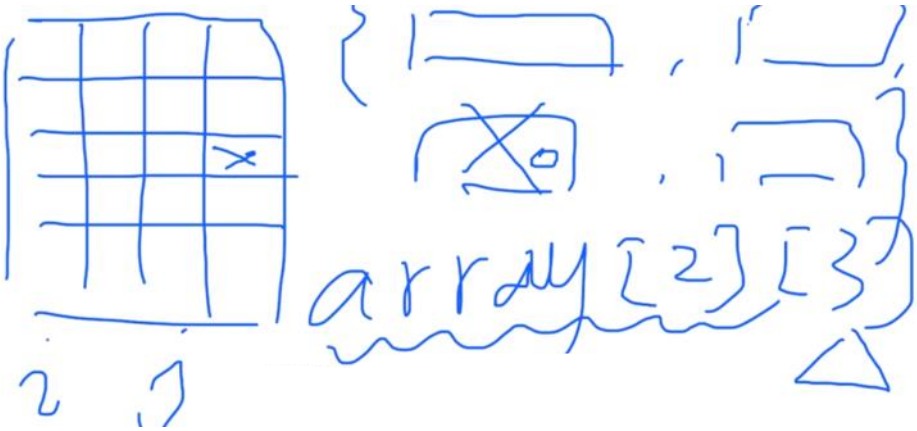
2.1.2 数组元素的查找

- 常见操作：查找、插入、删除
- “查找”包括：
  - 找x的序号（是否包含x）
  - 找第k个元素
  - 找max / min
  - 统计个数

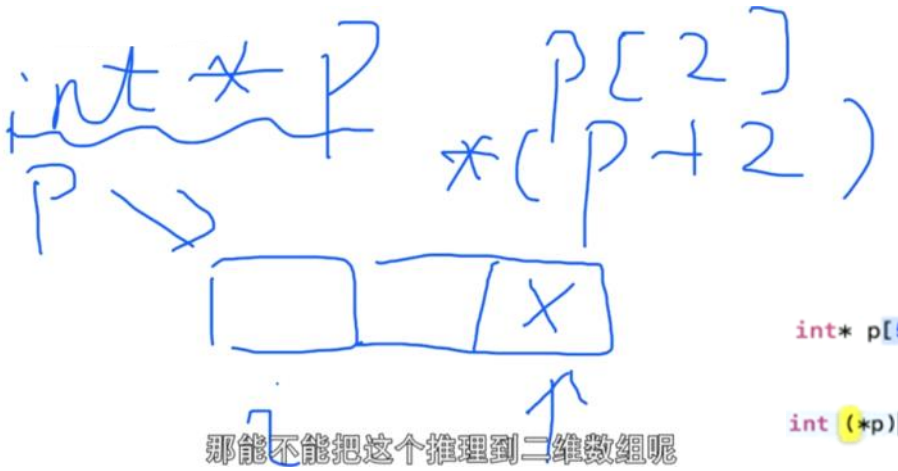
```c
#include <stdio.h>
#include <stdlib.h>

int findX(int* array, int size, int x){
    int flag = 0;
    int index = -1;
    for(int i=0; i<size; i++){
        if(array[i] == x) {
            flag = 1;
            index = i;
        }
    }
    return index;

}

int getX(int* array, int size, int k, int *px){
    if(k<0 || k>=size) return 0;
    else{
        *px = array[k];
        return 1;
    }
}
int main(int argc, const char * argv[]) {
    int array[10] = {111,22,3333,44,55};

    int k = 3;
    int x, flag;
    flag = getX(array, 10, k, &x);
    printf("%d, %d\n", flag, x);
```

2.2.1 二维数组的定义与使用

- 二维数组的实质：数组的数组
- 逻辑上可看作二维，其实并不是“二维”
- 怎样定义、使用？ C / C++ / Java / Python



array里面包含几个小数组
数组第二个元素是一个数组，里面找第三号元素



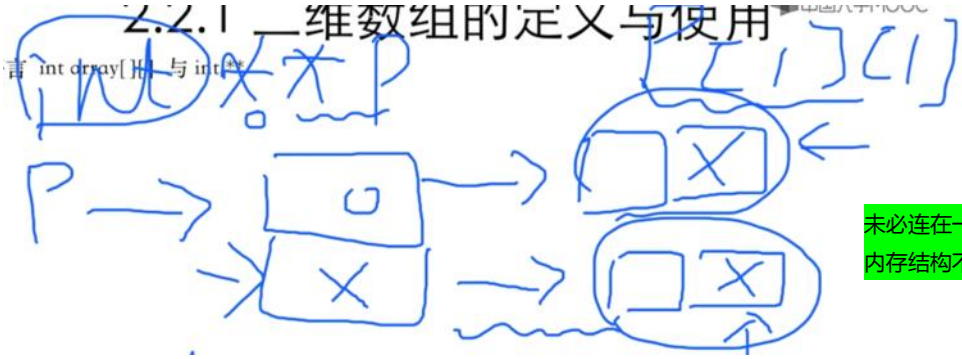
可以把一维数组和指针联系起来
*(p+2)连续往后找两个，类似数组
但是二维数组不可以！！

```
int* p[5];
```

看中心点，p开始往两边看，括号更优先，
p类型是一个数组，数组里面元素是指针

```
int (*p)[5];
```

p离*最近，是指向数组的指针
数组五个元素，每个都是int




```
int (*p) [5];

p = array;
```

未必连在一起，动态内存分配时可能相隔很远
内存结构不同

```
void func(int array[][5], int k){

}
int main(int argc, const char * argv[]) {
    int array[3][5];

    int (*p) [5];
    p = array;

    func(array, 3);
```

扔进去了array，值传递传指针，
array[][5]也是一个指针（值）

```
void func(int (*) [5], int);
void func(int (*p)[5], int k){

}
```

声明这么写，虽然看上去有点奇怪

```
void func(int (*p)[5], int k){

}
int main(int argc, const char * argv[]) {
    int array[3][5];

    int (*p) [5];
    p = array;

    func(array, 3);
```

2.2.2 二维数组实例：大炮打蚊子

现在，我们用大炮来打蚊子：蚊子分布在一个$M \times N$格的二维平面上，每只蚊子占据一格。向该平面的任意位置发射炮弹，炮弹的杀伤范围如下示意：

```
 o
oxo
 o
```

其中，`x`为炮弹落点中心，`o`为紧靠中心的四个有杀伤力的格子范围。若蚊子被炮弹命中（位于`x`格），一击毙命，若仅被杀伤（位于`o`格），则损失一半的生命力。也就是说，一次命中或者两次杀伤均可消灭蚊子。现在给出蚊子的分布情况以及连续`k`发炮弹的落点，给出每炮消灭的蚊子数。

输入格式:

第一行为两个不超过20的正整数`M`和`N`，中间空一格，表示二维平面有`M`行、`N`列。

接下来`M`行，每行有`N`个`0`或者`#`字符，其中`#`表示所在格子有蚊子。

接下来一行，包含一个不超过400的正整数`k`，表示发射炮弹的数量。

最后`k`行，每行包括一发炮弹的整数坐标`x`和`y`（$0 \le x < M$，$0 \le y < N$），之间用一个空格间隔。

输出格式:

对应输入的`k`发炮弹，输出共有`k`行，第`i`行即第`i`发炮弹消灭的蚊子数。

二维数组当一维存放，如果不检查是否越界，会跑到前一行最后一个里面去

```
int main(int argc, const char * argv[]) {

    scanf("%d%d", &M, &N);
    for(int i=0; i<M; i++){
        for(int j=0; j<N; j++){
            board[i][j] = getchar() == '0' ? 0 : 2;
        }
    }

    for(int i=0; i<M; i++){
        for(int j=0; j<N; j++){
            printf("%d", board[i][j]);
        }
        putchar('\n');
    }

    return 0;
}
```

```
5 6
 00#00#
 000###
 00#000
 000000
 00#000
220020
022200
022222
002000
```

5,6 后面的换行符留在了缓冲区！！
换行符不是0，给了2的值
每一行后面依然有换行符
而且题目前面会有空格（删去！）

```
#include <stdio.h>
int board[20][20];
int M,N;
int bang(int x, int y, int kill){
    if( (x>=0 && x<M) && (y>=0 && y<N) && board[x][y]>0 ){
        board[x][y] -= kill;
        if(board[x][y]<=0) return 1;
    }
    else{
        return 0;
    }
}

scanf("%d%d", &M, &N);
getchar();
for(int i=0; i<M; i++){
    for(int j=0; j<N; j++){
        board[i][j] = getchar() == '0' ? 0 : 2;
    }
    getchar();
}
int k;
scanf("%d", &k);
for(int i=0; i<k; i++){
    int x, y;
    scanf("%d%d", &x, &y);
```

2.3.1 字符串的定义与使用

C语言

- "abcdefg" 或 字符数组
- 字符串 和 字符数组 不同
- 注意安全性

```
    int count;
    count = 0;
    count += bang(x, y, 2);
    count += bang(x-1, y, 1);
    count += bang(x+1, y, 1);
    count += bang(x, y-1, 1);
    count += bang(x, y+1, 1);
    printf("%d\n", count);
}

//    for(int i=0; i<M; i++){
//        for(int j=0; j<N; j++){
//            printf("%d", board[i][j]);
//        }
//        putchar('\n');
//    }

return 0;
```

2.3.2 字符串的常用操作

- 比较、拼接、复制
- 子串：截取、查找（定位）、替换、计数
- 匹配

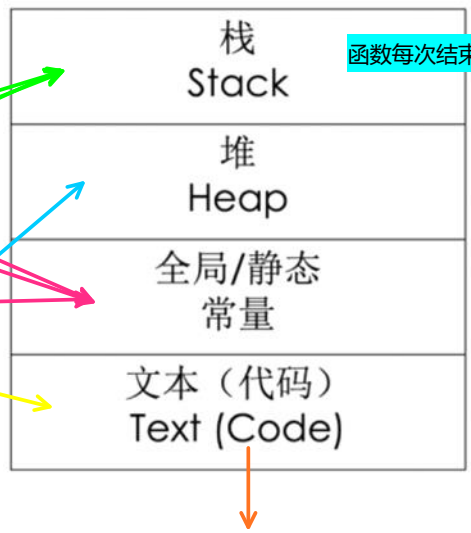
C中字符串不可变，字符串放在常量区
str,s2两个变量，放地址
s[]放栈区

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
    char *str = "hello world";
    printf("%p\n", str);
    char *s2 = "hello world";
    printf("%p\n", s2);
    return 0;
}
```

"hello word"放不进机器指令里面！放不下，就放进了常量区
malloc为何在堆里面?
1.跨函数使用
2.动态分配（堆负责动态分配）

```
0x100000fa2
0x100000fa2
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int total=0;
4  void hehe(){
5      static int he=0;
6      he++;
7      total++;
8  }
9  int main() {
10     int k=3;
11     char *str = "Hello, world!";
12     int *p = (int*)malloc(sizeof(int));
13     hehe();
14     printf("%d%s%d", k, str, *p);
15     free(p);
16     return 0;
17 }
```



函数每次结束，栈空间释放

编译后的二进制码

C语言程序执行的过程

✓编译（Compile）

源程序->目标代码

✓连接（Link）

一堆目标代码->程序

✓载入、执行（Load & Execute）

程序->进程

3个班进教室听课，每个班30个人。

编译：每个班内部序号1-30

连接：张三班里5号。连接过程：每个班进教室，2班先进，1-30，1班其次，31-60.。。。。。

再看：静态和动态

✓静态：在编译（编译+连接）阶段

✓动态：在执行阶段

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int m, n;
    int *p, *q;
    scanf("%d%d", &m, &n);
    p = (int*)malloc(sizeof(int) * m);
    q = (int*)malloc(sizeof(int) * n);
    return 0;
}
```

进一步：静态和动态 + static关键字

```c
#include <stdio.h>
#include <stdlib.h>
void hehe(){
    int k=0;
    static int sum=0;
    k++;
    sum++;
    printf("%d %d\n", k, sum);
}
int main(){
    int *p;
    p = (int*)malloc(sizeof(int));
    hehe();
    hehe();
    return 0;
}
```

static与它分配内存是动态还是静态无关！

C语言如何处理“类”和“对象”

- 没有“类”，有“类型”
- 没有“对象”，有“变量”
- 结构体变量 + 函数

```c
#include <stdio.h>
struct Student{
    int sid;
};
void sayId(const struct Student *ps){
    printf("My id is %d\n", ps->sid);
}

int main(){
    struct Student zs;
    zs.sid = 1001;
    sayId(&zs);
    return 0;
}
```

C语言如何处理“类”和“对象”

- 没有“类”，有“类型”
- 没有“对象”，有“变量”
- 结构体变量 + 函数
- 进一步：函数指针

```c
#include <stdio.h>
struct Student{
    int sid;
    void (*sayId)(const struct Student *);
};
void sayId(const struct Student *ps){
    printf("My id is %d\n", ps->sid);
}

int main(){
    struct Student zs;
    zs.sid = 1001;
    zs.sayId = sayId;
    zs.sayId(&zs);
    return 0;
}
```