9.2-1 指针是可计算的

1+1=2?

- 给一个指针加1表示要让指针指向下一个变量

  int a[10];

  int *p = a;

  *(p+1) —> a[1]

- 如果指针不是指向一片连续分配的空间，如数组，则这种运算没有意义

指针计算

- 这些算术运算可以对指针做:
  - 给指针加、减一个整数(+, +=, -, -=)
  - 递增递减(++/—)
  - 两个指针相减

*p++

- 取出p所指的那个数据来，完事之后顺便把p移到下一个位置去
- *的优先级虽然高，但是没有++高
- 常用于数组类的连续空间操作
- 在某些CPU上，这可以直接被翻译成一条汇编指令

指针比较

- <, <=, ==, >, >=, != 都可以对指针做
- 比较它们在内存中的地址
- 数组中的单元的地址肯定是线性递增的

0地址

- 当然你的内存中有0地址，但是0地址通常是个不能随便碰的地址
- 所以你的指针不应该具有0值
- 因此可以用0地址来表示特殊的事情：
  - 返回的指针是无效的
  - 指针没有被真正初始化（先初始化为0）
- NULL是一个预定定义的符号，表示0地址
  - 有的编译器不愿意你用0来表示0地址

指针的类型

- 无论指向什么类型，所有的指针的大小都是一样的，因为都是地址
- 但是指向不同类型的指针是不能直接互相赋值的
- 这是为了避免用错指针

指针的类型转换

- void* 表示不知道指向什么东西的指针
  - 计算时与char*相同（但不相通）
- 指针也可以转换类型
  - int *p = &i; void*q = (void*)p;
- 这并没有改变p所指的变量的类型，而是让后人用不同的眼光通过p看它所指的变量
  - 我不再当你是int啦，我认为你就是个void!

用指针来做什么

- 需要传入较大的数据时用作参数
- 传入数组后对数组做操作
- 函数返回不止一个结果
  - 需要用函数来修改不止一个变量
- 动态申请的内存...

9.2-2 动态内存分配

输入数据

- 如果输入数据时，先告诉你个数，然后再输入，要记录每个数据
- C99可以用变量做数组定义的大小，C99之前呢?
- int *a = (int*)malloc(n*sizeof(int));

malloc

```
#include <stdlib.h>

void* malloc(size_t size);
```

- 向malloc申请的空间的大小是以字节为单位的
- 返回的结果是void*，需要类型转换为自己需要的类型
  - (int*)malloc(n*sizeof(int))

没空间了?

- 如果申请失败则返回0，或者叫做NULL
- 你的系统能给你多大的空间?

free()

- 把申请得来的空间还给“系统”
- 申请过的空间，最终都应该要还
  - 混出来的，迟早都是要还的
- 只能还申请来的空间的首地址
- free(0)?

常见问题

- 申请了没free—>长时间运行内存逐渐下降
  - 新手：忘了
  - 老手：找不到合适的free的时机
- free过了再free
- 地址变过了，直接去free

以下内容不制作视频

9.2-3 函数间传递指针

好的模式

- 如果程序中要用到动态分配的内存，并且会在函数之间传递，不要让函数申请内存后返回给调用者
- 因为十有八九调用者会忘了free，或找不到合适的时机来free
- 好的模式是让调用者自己申请，传地址进函数，函数再返回这个地址出来

```c
int* init(int a[], int length);
int* print(int a[], int length);

int main()
{
    const int MAX_SIZE = 1000;
    int size;
    do {
        printf("输入数量(0,1000)：");
        scanf("%d", &size);
    } while (size>0 && size<MAX_SIZE);
    int* a = (int*)malloc(size*sizeof(int));
    print(init(a, size), size);
    free(a);

    return 0;
}
```

```c
int* init(int a[], int length)
{
    int i;
    for ( i=0; i<length; i++ ) {
        a[i] = i;
    }
    return a;
}

int* print(int a[], int length)
{
    int i;
    for ( i=0; i<length; i++ ) {
        printf("%d\t", a[i]);
    }
    printf("\n");
    return a;
}
```

除非函数的作用就是分配空间，否则不要在函数中malloc然后传出去用

在
同一个地方malloc和
free

函数返回指针?

- 返回指针没问题，关键是谁的地址?
  - 本地变量（包括参数）？函数离开后这些变量就不存在了，指针所指的是不能用的内存
  - 传入的指针？没问题
  - 动态申请的内存？没问题
  - 全局变量—>以后再解释

函数返回数组?

- 如果一个函数的返回类型是数组，那么它实际返回的也是数组的地址
- 如果这个数组是这个函数的本地变量，那么回到调用函数那里，这个数组就不存在了
- 所以只能返回
  - 传入的参数：实际就是在调用者那里
  - 全局变量或动态分配的内存

和返回指针是一样的!