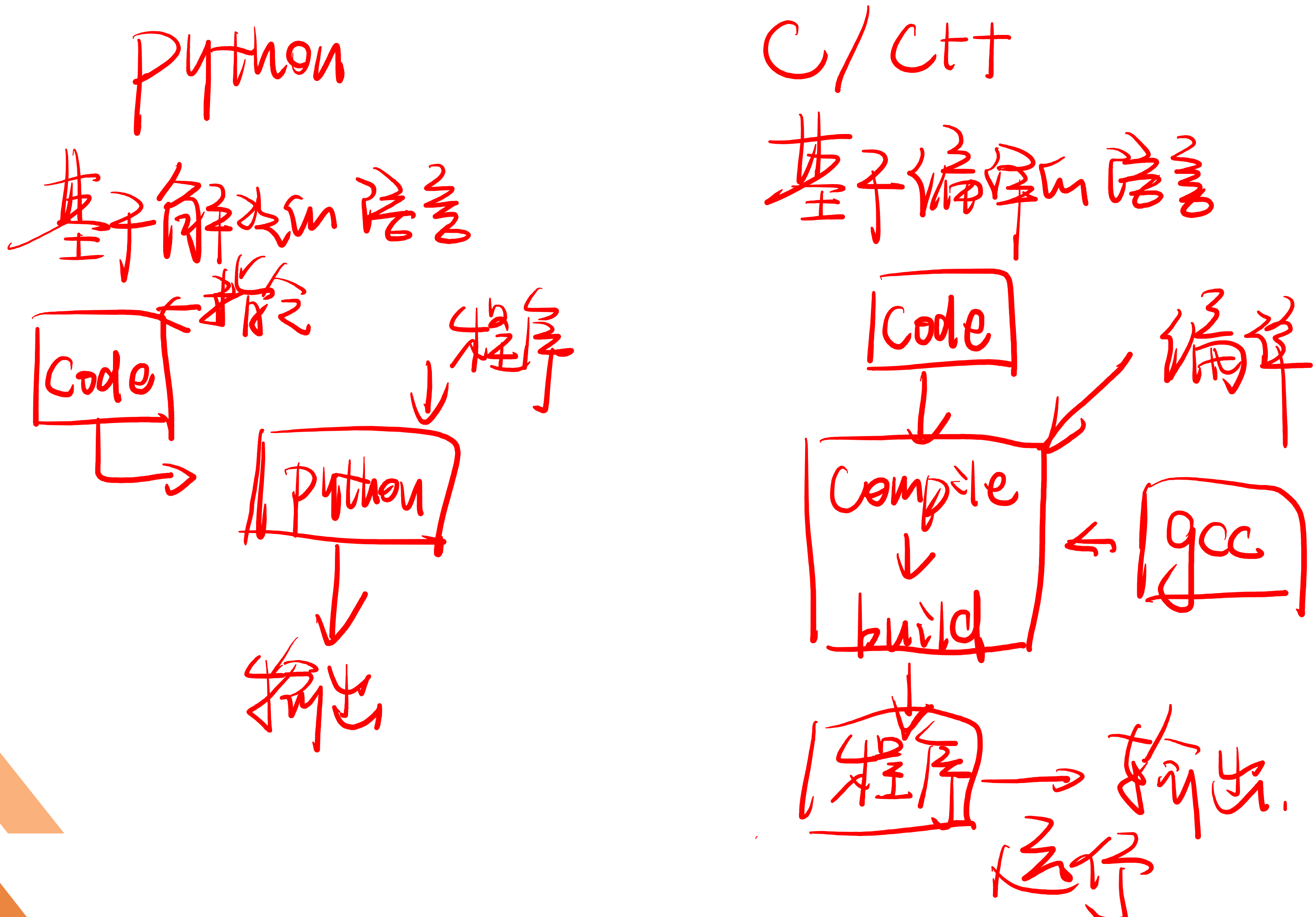
数据结构与算法

计算机学院

朱晨阳 副教授

zhuchenyang07@nudt.edu.cn

程序


Python
基于解释的语言
指令
Code
Python
程序
输出
C/C++
基于编译的语言
Code
编译
Compile
build
gcc
程序
输出
运行

数据

(硬件)

存储数据

程序

算法数据

list结构

c = [0]*10

编译

int c[10];

10位的int数组

栈区(Stack)

堆区(heap)

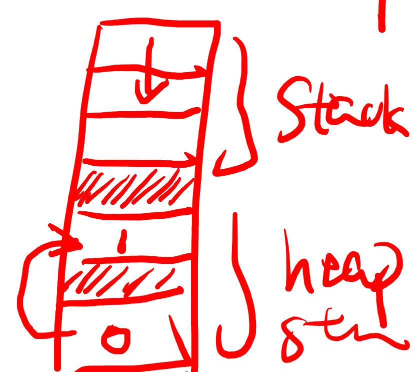
int *c = (int*) malloc(10*sizeof(int))

new

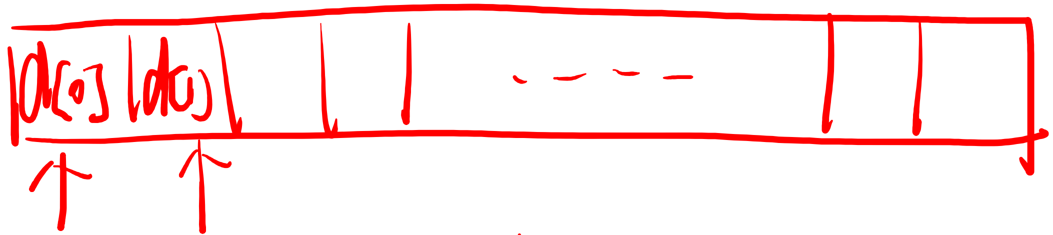
运行

连续存放的，大小是有限制的

不是连续，大小一般无限制



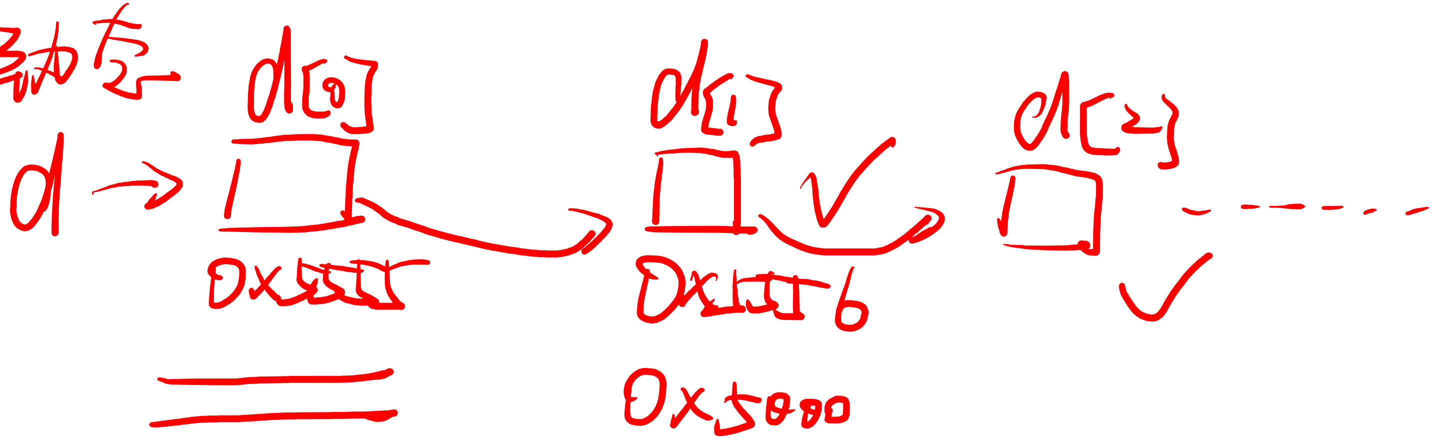
静态 → 10



预留

0x5555 0x5556 连续

动态



预留

什么是算法

- 算法是一个有穷指令序列。
- 算法的描述方式包括：程序语言、自然语言、图形方式、表格方式等。

什么是算法

- 一个算法具有以下特性：
- 输入
- 输出
- 确定性
- 有穷性
- 可行性

A = function(B,C)

A=B+C　　B+C等于A的一部分

while（1）{}

A=B/0

什么是好的算法

- 通常从以下几个方面衡量算法的好坏
  - 正确性
  - 可读性
  - 健壮性
  - 时间复杂度
  - 空间复杂度

```
int asdfghjkl[10];
int x[1000000000];
```

可不可以直接用一个算法运行了多久来衡量复杂度?

语句执行时间之和

- 算法的运行时间是该算法各条语句执行时间之和，每条语句的执行时间是该语句的执行次数与执行一次所需时间之积。
- 一个算法在计算机上运行消耗的时间取决于多种因素：算法的策略、求解问题的规模、程序设计语言的级别、编译程序的优劣、机器运行的速度。

时间复杂度

- 衡量算法的时间效率应该独立于计算机的软、硬件因素。
- 算法的时间复杂度是指算法中各条语句的执行次数之和。复杂度通常是问题规模n的函数，记为T(n)。

时间复杂度

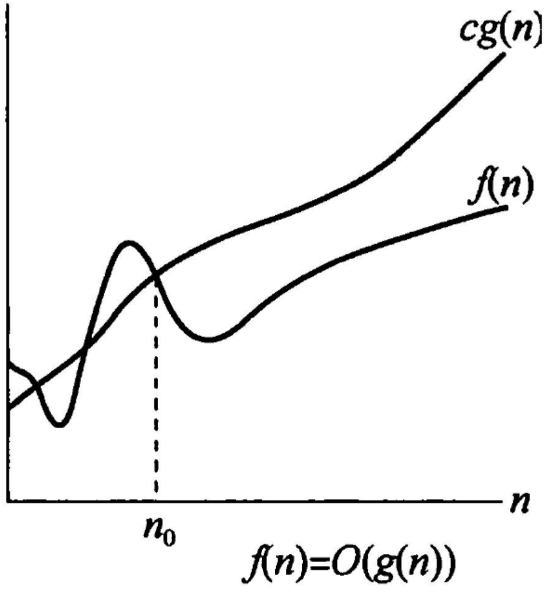
- 时间复杂度的精确表示经常是困难的，因而通常采用渐进时间复杂度。
- **T(n)** $\in$ **f(n)**，**T(n)**为算法的时间复杂度，**O(f(n))**表示其渐进时间复杂度——往往简称为时间复杂度。

大 O 符号

- 定义 设 f 和 g 是定义域为自然数集 N 上的函数，若存在正数 c 和 n0，使得对一切 n ≥n0 有

$$0 \le f(n) \le cg(n)$$

- 成立，则称 f(n) 的渐近的上界是 g(n)，记作
- f (n) = O(g(n))



大 O 符号

- n趋向无穷大时，f(n)/g(n)为正的常数。
  - 例如，$3n^3 + 2n + 1 \in O(\ ?\ )$ n^3
- 最坏情况下的时间复杂度、最好情况下的时间复杂度、平均时间复杂度

例子

```
void MatrixMultiply(int A[n][n], int B[n][n], int C[n][n])
{
        int i,j,k;
        for(i=0; i<n; i++)                          //(1)
            for(j=0; j<n; j++)                      //(2)
            {
                C[i][j]=0;                          //(3)
                for(k=0; k<n; k++)                  //(4)
                    C[i][j]=C[i][j]+A[i][k]*B[k][j] //(5)
            }
}
```

(1) $n+1$

(2) $n(n+1)$

(3) n^2

(4) $n^2(n+1)$

(5) n^3

T(n)= $2n^3+3n^2+2n+1 \in O(n^3)$

习题1.6(1)(2)

①

```
int i,k;
i=1;
k=0;
while(i<n)
{
    k=k+10*i;
    i++;
}
```

②

```
int i,k;
i=0;
k=0;
do
{
    k=k+10*i;
    i++;
}while(i<n);
```

习题1.6(3)(5)

③

```
int i,j;
i=1;
j=0;
while(i+j<=n)
{
    if(i>j) j++;
    else i++;
}
```

⑤

```
int x,y;
x=91; y=100;
while(y>0)
    if(x>100)
    {
        x=x-10;
        y--;
    }
    else x++;
```

排序算法的效率

- 案例：
  - 计算机 **A**(每秒十亿)：插入排序，$c_1 n^2$
  - 计算机 **B**(每秒一千万)：归并排序， $c_2 nlogn$

设 $c_1=2$， $c_2=50$，对一百万个元素排序：

排序算法的效率

- 案例：
  - 计算机 **A**(每秒十亿)：插入排序，$c_1 n^2$
  - 计算机 **B**(每秒一千万)：归并排序， $c_2 nlogn$

设 c_1 =2， c_2 =50，对一百万个元素排序：

- 计算机 **A**：**2000** 秒； 计算机 **B**：**100** 秒

对一千万个元素排序：

- 计算机 **A**：**2.3** 天； 计算机 **B**：**50** 分钟

多项式函数与指数函数

时间复杂度函数	问题规模					
	10	20	30	40	50	60
n	10^{-5}	$2*10^{-5}$	$3*10^{-5}$	$4*10^{-5}$	$5*10^{-5}$	$6*10^{-5}$
n^2	10^{-4}	$4*10^{-4}$	$9*10^{-4}$	$16*10^{-4}$	$25*10^{-4}$	$36*10^{-4}$
n^3	10^{-3}	$8*10^{-3}$	$27*10^{-3}$	$64*10^{-3}$	$125*10^{-3}$	$216*10^{-3}$
n^5	10^{-1}	3.2	24.3	1.7 分	5.2 分	13.0 分
2^n	.001 秒	1.0 秒	17.9 分	12.7 天	35.7 年	366 世纪
3^n	.059 秒	58 分	6.5 年	3855 世纪	$2*10^8$ 世纪	$1.3*10^{13}$ 世纪

基本函数类

- 阶的高低
- 至少指数级：$O(2^n), O(3^n), O(n!), \ldots$
- 多项式级：$O(n), O(n^2), O(nlogn), O(n^{\frac{1}{2}}), \ldots$
- 对数多项式级：$O(logn), O(log^2n), \ldots$
- 常数时间：$O(1)$