数据结构 第一章 数据结构基本认知和前置知识点复习

前言：

第一章 数据结构基本认知和前置知识点复习

前言：

*这份笔记，是我在学习数据结构过程中记下的笔记，并**未完全遵循**专业课的既定顺序与讲授逻辑。*

*于数据结构而言，这门学科**从无捷径可走** —— 若想真正学懂吃透，核心唯有“**多练**、**多敲**”。在此必须坦诚：我并**不认同**所谓的“四件套速通课”，这类课程往往只能带你浅尝皮毛，难以触及知识的核心与本质。*

*如果你想要紧跟课堂进度，那么匹配课程的幻灯片会是更直接的参考（当然，前提是你能沉下心去研读）。不过市面上多数数据结构相关的书籍与幻灯片，存在一个共性问题：**伪代码**占比过高。若盲目照着这些伪代码直接编写，最终能成功运行的概率恐怕不足 10%。*

*正因如此，若你不嫌弃这份笔记的随性与朴实，我十分欢迎你将其作为学习参考。希望这些基于我个人实践的记录，能帮你更轻松地理解数据结构的逻辑，辅助你的学习之路。实现代码分为**C语言描述**与**C++语言描述**，各位按需获取。*

最后说下这份笔记的使用建议：文中并未设置严格的章节划分，你大可以像读小说一样轻松浏览。唯独遇到具体的代码实现部分时，建议你停下脚步 —— 先完整读懂代码逻辑，再关掉笔记，试着独立敲写出属于自己的版本。唯有亲手实践，才能将知识真正内化为自己的能力。

By：ItsJiale

2025.10.1

第一章 数据结构基本认知和前置知识点复习

Q：什么是数据结构?

A：

在一些特定的、特殊的需求场景下，我们以往所学的数据类型，无法基于需求合理地组织数据，此时就需要我们自己设计一套新的数据组织形式来解决问题，也就是数据结构。

数据结构是一种存储、组织数据的方式，旨在便于访问和修改。没有一种单一的数据结构对所有用途均有效，所以需要知道几种数据结构优势和局限。

算法（Algorithms）+数据结构（Data Structures）=程序（Programs）

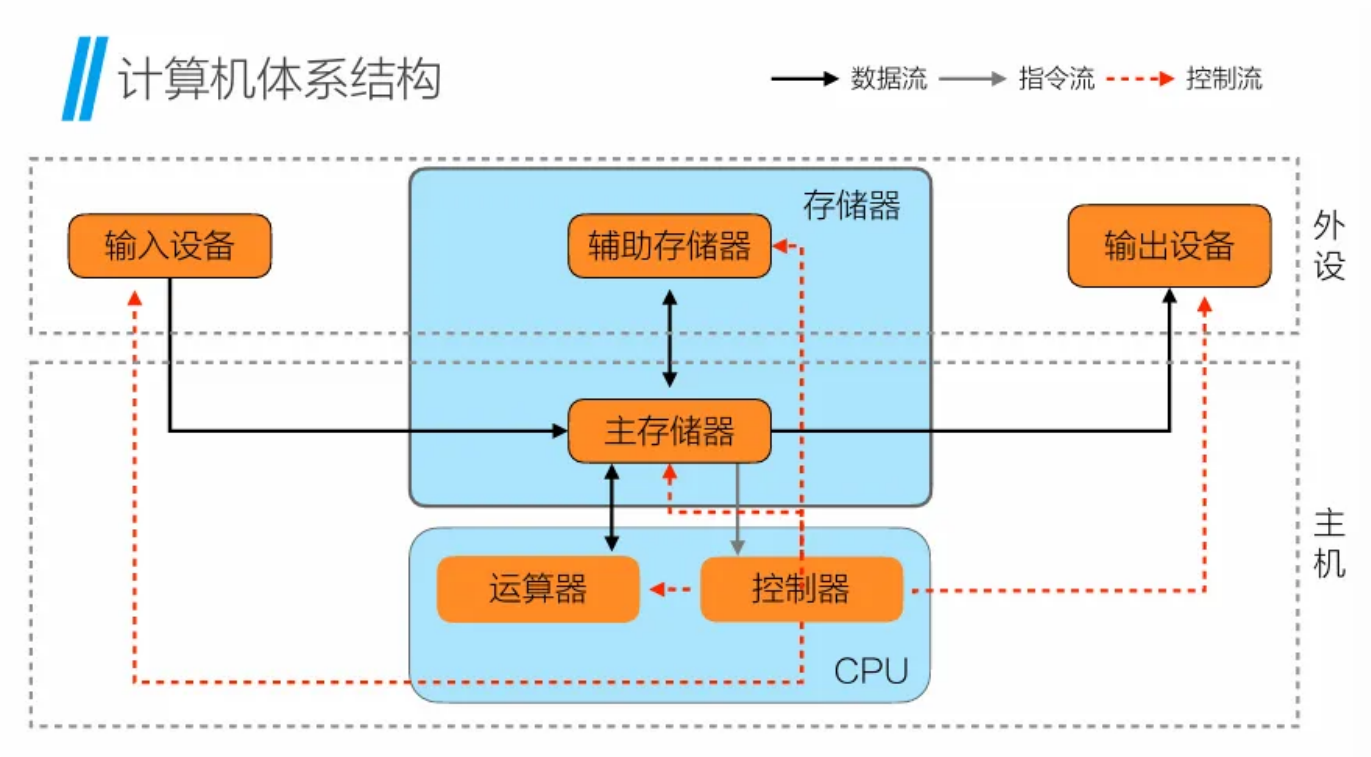
Q：什么是算法?

A：算法是一个有穷规则的集合，它用规则规定了解决某一特定类型问题的运算序列，或者规定了任务执行或问题求解的一系列步骤。

Q：基于 C 语言描述的数据结构需要哪些前置知识?

A： 函数 数组 字符串 指针 内存解析 结构体

由于函数、数组、字符串都比较熟悉，所以略过，但是要注意的是 C语言中**没有string**这个数据结构，而是采用char数组遍历的方式实现字符串



虚拟内存地址：

内存条、显卡、各种适配卡都有其各自的存储地址空间。

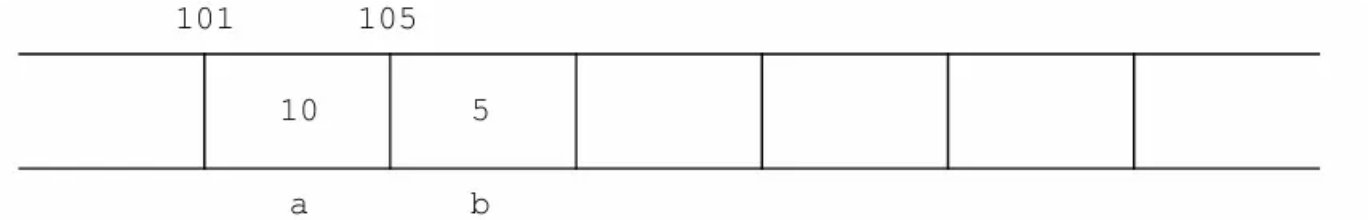
操作系统将这些设备的存储地址空间抽象成一个巨大的一维数组空间。

对于内存的每一个字节会分配一个 32 位或 64 位的编号，这个编号称为内存地址。

例如：

```C++
int main()
{
    int a = 10;
    int b = 5;
    return 1;
}
```

代码一旦运行则会在虚拟内存中开辟101和105的虚拟内存地址



虽然数组在已经接触了无数次，但是还是提一嘴：

使用取地址符 & 获取数组的地址时，返回的是数组第 0 个元素的内存地址

```C++
int main()
{
    int a[] = {16,47,89,42,38};
    printf("%p\n", &a);
    printf("%p\n", &a[0]);
    //二者是一致的
}
```

在学习DSA时，指针和结构体是无可避免的，DSA**学的好坏**取决于对指针的理解程度

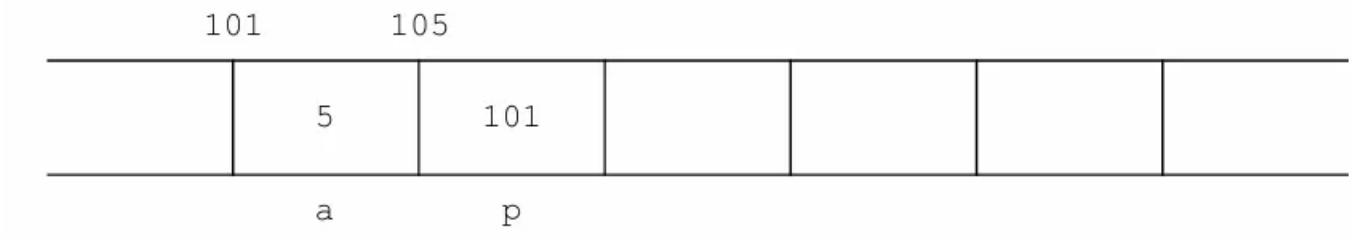
指针：指针是用来**存放内存地址的变量**

指针的声明如下：

```C++
int a;
//声明一个整型变量
int *p;  //声明一个指针变量，该指针指向一个int类型值的内存地址
//星号 * 两边的空格无关紧要，下面的声明是等价的
int* p;
int *p;
int * p;
int*p;
```

指针的使用如下

```C++
int a;
int *p;
a = 5;
p = &a;
```



间接引用操作符 *

间接引用操作符 * **返回**指针变量的**指向地址的值**

通常把这个操作叫做“解引用指针”

```C++
int a = 5;
int *p = &a;
printf("%d\n", *p);
*p = 100;
printf("%d\n", a)
```

指针做算术运算

给指针加上一个整数，实际上加的是这个整数和指针数据类型对应字节数的乘积

```c
int a = 5;
int *p = &a;
printf("%p\n", p);
p++;
printf("%p\n", p);
return 0;
```

p++ ——>p = p + 1——>p = p + 1 x 4

各数据类型字节数

有符号数	无符号数	32位	64位
char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned int	4	4
long	unsigned long	4	8
float		4	8
double		8	8

Q：为什么需要结构体?

A：因为C语言里没有类，不是面向对象编程，无法类的实现封装，可以粗略的等同于用结构体来代替类（Go语言里也是类似如此，没有类，用结构体来代替相关类的功能）

结构体的声明

结构体是一个或多个变量的集合，这些变量可以是不同的类型

```C++
struct 结构体名
{
数据类型 变量名;
数据类型 变量名;
    ……
};

struct point
{
    int x;
    int y;
};
```

结构体的初始化与调用

```C++
struct 结构体名 变量名;
struct point p;
变量名.结构体内部变量名 = 值
p.x = 10;
p.y = 15;
```

具体实现

```C++
struct point createPoint(int x, int y)
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

解读：传入两个int参数，在函数创建一个结构体point类型的变量， 将传入的两个参数分别赋值给该结构体变量的x和y，最后将该结构体变量返回

结构体与指针

在一些场景中，如果传递给函数的结构体很大，使用指针方式的效率通常更高。

pp指向一个point结构体

```C++
struct point *pp;
```

为pp所指向的结构体中的属性赋值

```C++
(*pp).x = 10;
(*pp).y = 5;
```

注意：！！！！

为了使用方便，C 语言提供了另外一种简写方式

```C++
pp->x = 10;
pp->y = 5;
```

这就是箭头的由来！！！一定要有印象并且熟记

类型定义

```C++
typedef 数据类型 别名
typedef int zx
```

解读：给int取一个别名叫做zx，以后int a = 10;等价于 zx a = 10;

```c
struct point
{
    int x;
    int y;
};

int main(int argc, char const *argv[])
{
    struct point p;
    p.x = 5;
    p.y = 10;
    printf("%d\n", p.x);
    printf("%d\n", p.y);
    return 0;
}
```

每次声明结构体变量都要写 struct 关键 字，很麻烦，而且逻辑上也很难受，typedef 可以解决这个问题。 以下两种方式即可：

C++

```c
typedef struct 结构体名
{
数据类型 变量名;
数据类型 变量名;
    ……
}别名;

typedef struct point
{
    int x;
    int y;
}po;
```

C++

```c
typedef struct
{
数据类型 变量名;
数据类型 变量名;
    ……
}别名;

typedef struct
{
    int x;
    int y;
}po;
```

内存分配

C 程序编译后，会以三种形式使用内存

静态 / 全局内存：

静态声明的变量和全局变量使用这部分内存，这些变量在程序开始运行时分配，直到程序终才消失。

自动内存（栈内存）：

函数内部声明的变量使用这部分内存，在函数被调用时才创建。

动态内存（堆内存）：

根据需求编写代码动态分配内存，可以编写代码释放，内存中的内容直到释放才消失。

动态内存分配

在C语言中，动态分配内存的基本步骤：

1. 使用 malloc （memory allocate）函数分配内存

void* malloc(size_t)

如果成功，会返回从堆内存上分配的内存指针

如果失败，会返回空指针

2. 使用分配的内存

3. 使用 free 函数释放内存

具体步骤我们**之后以实例讲解**

在C++语言中，动态内存分配使用new和delete且往往与析构函数有关