第十三讲

Pointers, Arrays, Structs

薛浩

2023 年 5 月 18 日

www.stickmind.com

今日话题

话题 1：编程基础 初学编程的新手，一般应该熟练使用函数和库处理字符串相关的编程任务。

话题 2：抽象数据类型的使用 在尝试实现抽象数据类型之前，应该先熟练使用这些工具解决问题。

话题 3：递归和算法分析 递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

话题 4：类和内存管理 使用 C++ 实现数据抽象之前，应先学习 C++ 的内存机制。

话题 5：常见数据结构 在熟练使用抽象数据类型解决常见问题之后，学习如何实现它们是一件很自然的事情。

话题 4：类和内存管理

学习使用 C++ 实现数据抽象之前，应先了解 C++ 的内存机制。

- 指针和数组
- 动态内存管理
- 类的设计

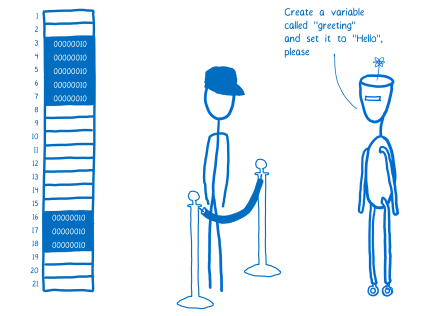


Figure 1: 数据封装和内存管理

计算机是如何存储信息的?

目录

复习：类型 Types

基本数据类型

掌握基本数据类型是学习一门编程语言的基础。C++ 有几种内置的类型，同时还允许程序员自定义数据类型。

```
char ch = 'F';
short val1 = 128;
int val2 = 32;
long val3 = 2022;
long long val4 = 7837432;

float decimalVal1 = 5.0;
double decimalVal2 = 5.0;

bool bVal = true;
```

字符串类型

很多现代编程语言都把字符串当作基本数据类型，但遗憾的是，C++ 并没有提供。
更混乱的是，C++ 中有两种不同的字符串风格，一种是 C 语言继承下来的旧式字符串，另一种是基于类的 string 类。

```
#include <string>
std::string name = "cs101";
```

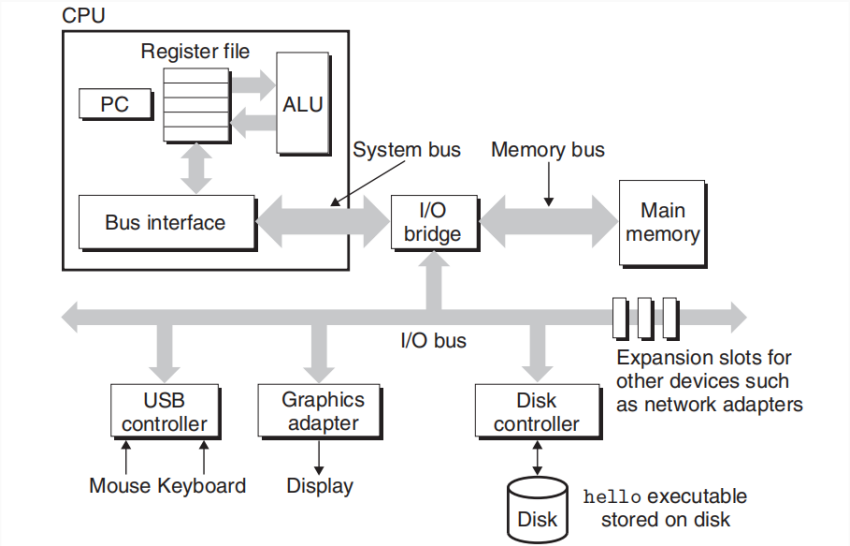
在 C++ 中，数据的类型必须明确定义，一旦变量被创建，其类型将无法改变。

练习：补全数据类型声明

```
_____ a = "test";
_____ b = 3.2 * 5 - 1;
_____ c = 5 / 2;
_____ d(int foo) { return foo / 2; }
_____ e(double foo) { return foo / 2; }
_____ f(double foo) { return int(foo / 2); }
_____ g(double c) { std::cout << c << std::endl; }
```

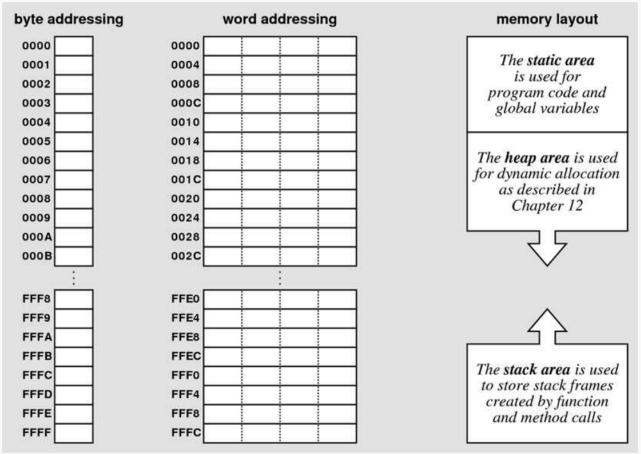
内存 Memory

计算机组成


CPU
Register file
PC
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters
USB controller
Graphics adapter
Disk controller
Mouse Keyboard
Display
Disk
hello executable stored on disk

内存抽象 Memory

内存以字节块为单位，每个位置有独特的编号称为地址，一般使用无符号整型的十六进制表示。



变量分配

声明一个变量时，编译器必须提供足够的内存空间存储该类型的值。

不同的数据类型需要的内存大小不同：

char	1 字节
bool	1 字节
int	4 字节
double	8 字节

指针 Pointer

指针 Pointer

指针（pointer）是一种特殊的类型，用来存储数据的内存地址，以便提供更灵活的数据访问方式。

指针变量存储的是内存某个位置的地址，所以指针变量的值本质上仍然是一个 64 位的整型。

指针提供了底层硬件的抽象机制和访问方式，通过指针可以直接处理内存中的字节。

指针声明和初始化

每个类型都有对应的指针类型，指针类型只需要在类型后加一个星号。

声明一个指针变量时，如果尚未确定存储某个数据的地址，可以使用特殊值 nullptr 表明指针为空。

```
int* iptr = nullptr;
char* cptr = nullptr;
GLine* gline = nullptr;
```

指针类型的宽度由硬件决定，64 位系统下指针占用的内存是 8 个字节。

指针基本运算

C++ 定义了两个指针运算符，允许在指针和数据之间进行运算：

`&` 获取地址

`*` 获取指针指向的值

指针的初始化和赋值只能使用有地址的表达式，这类值称为**左值**。

```
int x = 1, y = 2;
int* p1 = &x;
int* p2 = &y;
int* p3 = &(x + y); // Wrong
```

指针基本运算

操作符 * 返回的是指针所指变量的左值，所以使用上和变量一致。

```
int x = 1;
int* p = &x;
*p = 3; // x = 3;
```

指针 vs 引用

引用必须指向某个对象，而指针可以指向某个对象，也可以为 nullptr 表示空指针。

指针可以被重新赋值，而引用一旦创建则无法修改其引用的对象。

数组 Array

数组 Array

数组（Array）是一种较为低级的数据集合，概念上和 Vector 类似。

有序 可以像 Vector 一样，根据索引值依次访问元素

同质 数组中的每个元素必须是相同的类型

虽然数组在使用上和 Vector 类似，但两者的差别和 C 字符串和 string 类的差异很相似。

- 数组一旦分配好内存后，其大小无法改变
- 数组大小仅是概念上的大小，编译器不会阻止越界访问
- 数组没有类似 Vector 的接口，例如插入、删除等

数组声明和初始化

数组声明的一般形式如下，创建 10 个元素的 int 数组：

```
int arr[10];
```

数组也可以在声明时初始化：

```
string dir[] = {"East", "West", "South", "North"};
char str[] = "hello";
```

数组大小

由于编译器不会阻止越界访问，所以最好维护一个变量：

```
int nElems = 10;
int arr[nElems];
```

对于索引操作就可以做好防御检查：

```
int index = ...;
if (index >=0 && index < nElems) {
  arr[index];
}
```

数组大小

指定数组容量的大小称为**分配容量**；数组中实际存储的元素个数称为**有效容量**。

分配容量还可以通过 sizeof 计算得出：

```
int allocatedSize = sizeof arr / sizeof arr[0];
```

数组 vs 指针

数组的用法有时很像指针，但两者确实完全不同的类型。数组区别的指针的地方在于其包含的信息更为丰富：

地址 数组变量的表达式是数组首元素的内存地址

容量 通过数组变量可以计算出数组的分配容量

类型 数组元素的类型决定了数组最终的空间占用

从内存模型角度，很容易区分指针和数组。

结构体 Structure

结构体 Structure

结构体（Structure）用于组合不同数据类型，形成一个有意义的复合类型。

结构体的声明和初始化方式如下，通过点运算符访问结构体成员：

```
struct Point {
    int x;
    int y;
};
Point pt;
pt.x = 1;
pt.y = 2;
```

也可以使用列表初始化的方式：

```
Point pt = {1, 2};
```

结构体指针

结构体类型一旦定义，也对应一个结构体类型指针，可以通过指针操作访问结构体成员：

```
Point pt = {1, 2};
Point* p = &pt;
cout << (*p).x << endl;
cout << (*p).y << endl;
```

除此之外，还可以使用 -> 运算符：

```
cout << p->x << endl;
cout << p->y << endl;
```

计算机是如何存储信息的?