# 第十四讲

*Dynamic Allocation, LinkedList*

薛浩

2023 年 5 月 23 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

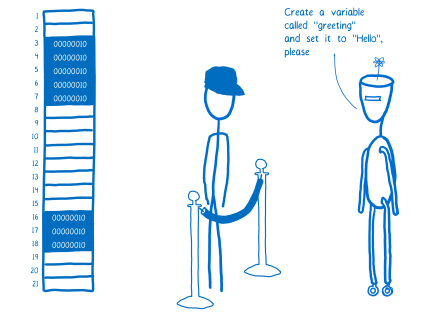


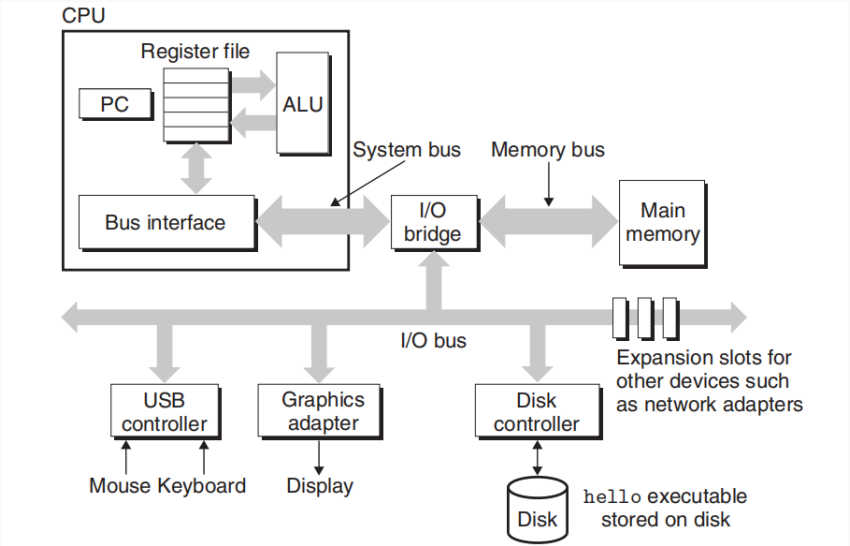

Figure 1: 数据封装和内存管理

# 计算机是如何分配内存的?

# 目录

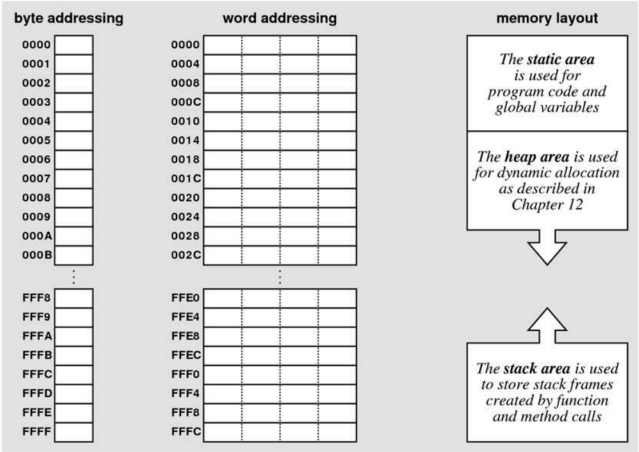
# 复习：内存和结构体

# 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 Memory

内存类似一个大号数组，由字节组成，每个字节唯一编号：



# 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。

声明一个结构体需要定义结构体的成员和成员类型：

```
struct Node {
    char ch;
    Node* next;
};
```

使用递归的方式定义的结构体可以实现一个非常重要的数据结构——链表！

# 动态内存分配

# 变量的内存分配模式

在程序中声明一个变量时，C++ 会在内存上为该变量分配一块内存。

**自动分配** 在函数栈上创建的局部变量，分配和释放由系统自动完成

**静态分配** 以 const 修饰的全局变量或 static 修饰的静态变量

**动态分配** 程序员创建并管理的，分配在堆上的变量或对象

# 动态内存分配

使用 new 运算符可以在堆上分配内存，其语法格式如下：

```
new Type(args);
```

该语法会使用 args 在堆上创建类型 Type 的对象，并返回一个指针。

```
int* p = new int;
Node* n = new Node;
n->ch = 'a';
n->next = nullptr;
```

# 动态内存释放

堆上分配的内存需要程序员释放，使用 delete 运算符可以释放内存，其语法格式如下：

```
delete p;
delete n;
```

注意，该语句只会释放堆上的内存，由于指针在栈上，所以并不会影响。

# 链表 LinkedList

# 链表 LinkedList

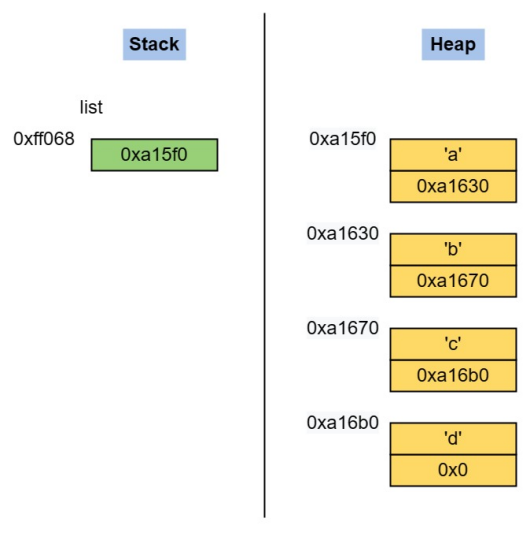
**链表**（LinkedList）是由节点（node）组成的链式数据结构。每个节点包含两个信息：

- 数据信息
- 指向下一个节点的指针

节点可以使用类似如下递归结构体来表示：

```
struct Node {
    char ch;
    Node* next;
};
```

# 链表 LinkedList



# 链表的表示

```
Node* list = new Node('a', nullptr);
list->next = new Node('b', nullptr);
list->next->next = new Node('c', nullptr);
list->next->next->next = new Node('d', nullptr);
```

# 遍历链表

```
void printList(Node* list) {
    while(list != nullptr){
        cout << list->ch << endl;
        list = list->next;
    }
}
```

# 释放链表

```
void cleanupList(Node* list) {
    while (list != nullptr) {
        Node* node = list;
        list = list->next;
        delete node;
    }
}
```

# 插入节点

```
void insertNode(Node*& list, char ch) {
    Node* node = new Node(ch, nullptr);
    node->next = list;
    list = node;
}
```

# 插入节点

```
void appendNode(Node*& list, char ch) {
    Node* node = new Node(ch, nullptr);
    if (list == nullptr) {
        list = node;
    } else {
        Node* tail = list;
        while (tail->next != nullptr) {
            tail = tail->next;
        }
        tail->next = node;
    }
}
```

# 计算机是如何分配内存的?