



中国科学技术大学
University of Science and Technology of China

数据结构

课程介绍

September 10, 2025

目录

① 课程基本信息

② 预修知识

课程基本信息

- 教材: 《数据结构》(c 语言版)(第二版), 严蔚敏等编著, 人邮出版社
- 参考书目
 - 《数据结构题集 (c 语言版)》, 严蔚敏等, 清华大学出版社
 - 《数据结构第 2 版》黄刘生、唐策善, 中国科技大学出版社
 - "Data Structures with C++" Williaaw Ford et al., Prentice Hall Inc.
 - "Data Structures Program Design in C, 2nd Ed." Robert Kruse et al., Prentice Hall Inc.
- 学时/学分: 理论课时 60; 实验课时 40; 学分 4
- 上课时间: 1-15 周, 周一下下午 (6, 7)、周三下午 (6, 7)
- 上课地点: 3C301
- 考核方式: 统一闭卷考试, 成绩计算: 作业 + 课堂学习笔记 + 上机 + 期末考试

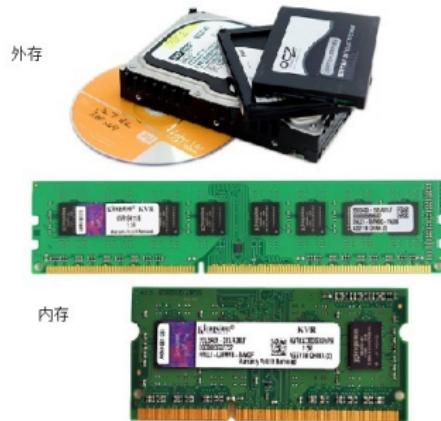
助教和课程 QQ 群

助教姓名	电子邮件	负责学生分组	备注
周驰斌		A	
张羽寒		B	
		C	
马筱雅		D	
陈彦羽		E	

课程 QQ 群号：595013400, 群名称:2025 年秋季数据结构

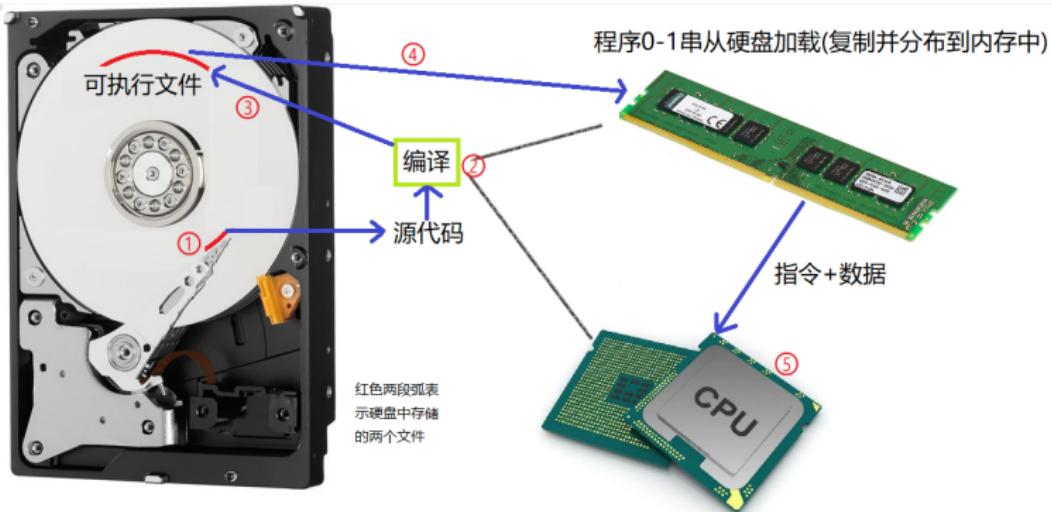
- 预修课程：C 程序设计语言
- 必需知识：
 - 熟练掌握 C 语言，用 C 语言编写、编译和调试简单程序的实践能力
 - 计算机组成和工作的基本原理

计算机组成



- 核心组件: CPU + 内存 = 指令执行器 + 工作场所
- 区别内存和外存 (硬盘、U 盘、光盘、磁盘、网盘等等)

计算机程序设计与运行



- ①: 编写程序源代码, 存入外存
- ②: 编译源代码
- ③: 保持可执行文件, 存入外存
- ④: 程序启动
- ⑤: 指令和数据读入 CPU 和寄存器, 执行

计算机工作的基本原理与编程

计算机工作原理的要点：

- 必不可少的部件：CPU 和内存
- 指令和数据都是 0-1 串，存放在内存中
- CPU 从内存中读取指令和数据，执行指令规定的操作和运算，并返回结果给内存（缓冲区）

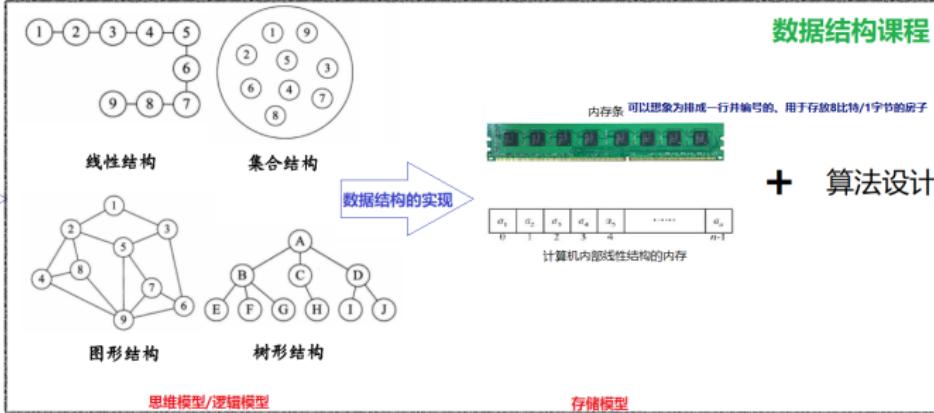
编程：将现实世界的事物及其处理过程变成内存中的 0-1 串

- 数据结构：将现实世界的事物映射成内存中的数据 0-1 串
- 算法：将数据处理过程描述出来，并由编译器转换为指令 0-1 串
- 编程 = 数据结构 + 算法

计算机程序设计进阶之路



建模

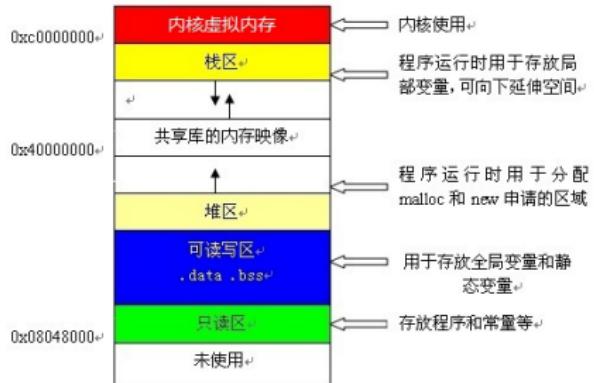


数据结构是基础

- 考虑程序设计中最基础的操作/运算 \Rightarrow 数据结构课程
- 考虑常见的算法 \Rightarrow 算法基础课程
- 考虑如何建模，如何求解非多项式时间复杂度能解的问题 \Rightarrow 人工智能课程

C 语言程序设计时与内存相关的关键概念

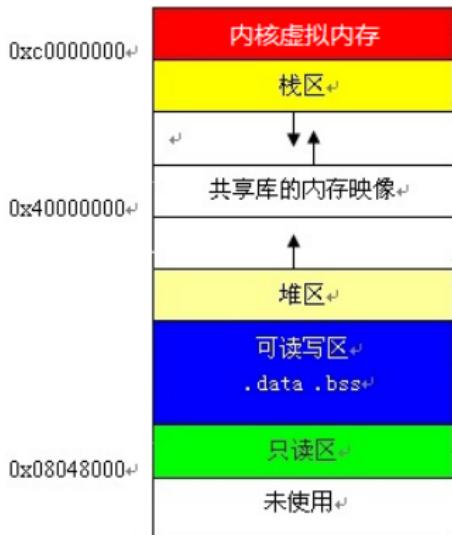
内核/系统区、栈、堆、可读写区和只读区



- 系统区：操作系统，直接访问或修改可能引发系统崩溃
- 栈：函数调用时的工作记录（包括函数内的局部变量，形参等），系统管理
- 堆：malloc() 或 c++ 中的 new 等操作分配的空间，程序员管理，可能会产生“碎片”，灵活但是效率低
- 可读写区：全局变量，静态变量等，系统管理
- 只读区：代码

C 语言程序设计时变量的存储位置

指出各种变量（数据）和指令在内存中的位置



- 指令：库函数 $\sin(x)$ 的执行指令，主函数 main 的指令，函数 fun1 的指令
- 变量：变量定义时的位置和类型决定了在程序加载到内存时的变量在内存的位置，填写下表

定义位置	int x	int *y	int *z=malloc
main 内			
一般函数内			
一般函数外			
形参			
一般函数内 static			
一般函数外 static			

理解变量的存储位置，是理解指针和函数调用时参数传递的关键！

C 语言程序中的参数传递 (1)



`main()` 函数加载执行期间，其内部定义的
变量 `x,y,z` 存在于栈区

```
1 int add(int a,int *b){  
2     a++; (*b)++;  
3     int c = a-*b;  
4     return c;  
5 }  
6 int w;  
7 int main(){  
8     int x=1;  
9     int *y=&x;  
10    int *z = (int  
11        *)malloc(sizeof(int));  
12    *z = 5;  
13    /* 程序执行到此，  
14    内存状态如左图 */  
15    *z += add(x,y);  
16    return *z;  
}
```

C 语言程序中的参数传递 (2)

0xC0000000	内核/OS
0xB000F020	$x = 1$ $y = 0xB000F020$ $z = 0x89210010$ $a = x = 1$ $b = y = 0xB000F020$ c ... (栈区)
0xA0000000	共享库
0x89210010	...
0x89200000	malloc() 分配空间, 5 ... (堆区)
0x08048000	可读写区
0	w 只读区 未使用

函数 add() 加载执行期间, 其形参 a,b 和内部定义的变量 c 存在于栈区

```
1 int add(int a, int *b){  
2     //程序执行到此,  
3     //内存状态如左图  
4     a++; (*b)++;  
5     int c = a+*b;  
6     return c;  
7 }  
8 int w;  
9 int main(){  
10    int x=1;  
11    int *y=&x;  
12    int *z = (int *)  
           malloc(sizeof(int));  
13    *z = 5;  
14    *z += add(x,y);  
15    return *z;  
16 }
```

C 语言程序中的参数传递 (3)

0xC0000000	内核/OS
0xB000F020	$x = 1$ $y = 0xB000F020$ $z = 0x89210010$ $a = 2$ $b = y = 0xB000F020$ c ... (栈区)
0xA0000000	共享库
0x89210010	...
0x89200000	malloc() 分配空间, 5 ... (堆区)
0x08048000	可读写区
0	w 只读区 未使用

函数 add() 加载执行期间，其形参 a,b 和内部定义的变量 c 存在于栈区

```
1 int add(int a,int *b){  
2     a++;  
3     //程序执行到此,  
4     //内存状态如左图  
5     (*b)++;  
6     int c = a+*b;  
7     return c;  
8 }  
9 int w;  
10 int main(){  
11     int x=1;  
12     int *y=&x;  
13     int *z = (int *)  
             malloc(sizeof(int));  
14     *z = 5;  
15     *z += add(x,y);  
16     return *z;  
17 }
```

C 语言程序中的参数传递 (4)

0xC0000000	内核/OS
0xB000F020	$x = 2$ $y=0xB000F020$ $z=0x89210010$ $a = 2$ $b = y=0xB000F020$ c ... (栈区)
0xA0000000	共享库
0x89210010	...
0x89200000	malloc() 分配空间, 5 ... (堆区)
0x08048000	可读写区
0	w 只读区 未使用

函数 add() 加载执行期间，其形参 a,b 和内部定义的变量 c 存在于栈区

```
1 int add(int a,int *b){  
2     a++;  
3     (*b)++;  
4     //程序执行到此,  
5     //内存状态如左图  
6     int c = a+*b;  
7     return c;  
8 }  
9 int w;  
10 int main(){  
11     int x=1;  
12     int *y=&x;  
13     int *z = (int *)  
             malloc(sizeof(int));  
14     *z = 5;  
15     *z += add(x,y);  
16     return *z;  
17 }
```

C 语言程序中的参数传递 (5)

0xC0000000	内核/OS
0xB000F020	$x = 2$ $y=0xB000F020$ $z=0x89210010$ $a = 2$ $b = y=0xB000F020$ $c = 4$... (栈区)
0xA0000000	共享库
0x89210010	...
0x89200000	malloc() 分配空间, 5 ... (堆区)
0x08048000	可读写区
0	w 只读区 未使用

函数 add() 加载执行期间，其形参 a,b 和内部定义的变量 c 存在于栈区

```
1 int add(int a,int *b){  
2     a++;  
3     (*b)++;  
4     int c = a+*b;  
5     //程序执行到此,  
6     //内存状态如左图  
7     return c;  
8 }  
9 int w;  
10 int main(){  
11     int x=1;  
12     int *y=&x;  
13     int *z = (int *)  
             malloc(sizeof(int));  
14     *z = 5;  
15     *z += add(x,y);  
16     return *z;  
17 }
```

C 语言程序中的参数传递 (6)

0xC0000000	内核/OS
0xB000F020	$x = 2$ $y=0xB000F020$ $z=0x89210010$
	... (栈区)
0xA0000000	共享库
	...
0x89210010	malloc() 分配空间, 9
0x89200000	... (堆区)
	可读写区
0x08048000	w
0	只读区
	未使用

函数 add() 结束返回后, 其形参 a,b 和内部
定义变量 c 从栈区删除

```
1 int add(int a,int *b){  
2     a++;  
3     (*b)++;  
4     int c = a*b;  
5     return c;  
6 }  
7 int w;  
8 int main(){  
9     int x=1;  
10    int *y=&x;  
11    int *z = (int *)  
           malloc(sizeof(int));  
12    *z = 5;  
13    *z += add(x,y);  
14    //程序执行到此,  
15    //内存状态如左图  
16    return *z;  
17 }
```

C 语言程序中的参数传递 (7)

C 语言程序中，参数传递方式只有一种：

- 程序运行时，实参和形参是不同的“物理实体”，存在与内存的不同位置
- 参数传递是把实参的值传给形参
- 即把实参变量在内存中的 0-1 串复制给另一个内存位置（形参变量）
- 形参和实参为什么要类型相同？

C 语言程序中指针使用典型错误

- initList1(ListNode *p) 中 p 是形参，形参在函数被调用和运行期间在栈区被分配了内存空间
- ptr1 是实参，在栈区有内存空间，能保存一个指针；调用时 ptr1 没有被初始化，即 ptr1 中的值（地址，指针）是随机数，该随机数被复制给了 p
- initList1() 执行期间，通过 malloc() 获得了一个新的地址/指针，指向一个已分配的结构体空间（在堆区），该地址被复制给了 p
- initList1() 执行完毕，p 被删除，但是 p 指向的已分配的内存块被系统登记已经占用，但是已经无法通过 p 找到该内存块（内存泄漏），也无法使用该内存块，等待系统重启，释放这种无效的内存占用
- 上述过程中 ptr1 的值从来没有被修改过，仅在 initList() 调用时，访问/读取过一次 ptr1 的值

```
1 typedef struct Lnode {  
2     int data;  
3     struct Lnode *next;  
4 } LinkNode, *Linklist;  
5  
6 LinkNode *initList(){  
7     LinkNode *p = (LinkNode *)  
8         malloc(sizeof(LinkNode));  
9     (!p)? exit(0):p->next=NULL;  
10    return p;}  
11  
12 void initList1(ListNode *p){  
13     p = (ListNode *)  
14         malloc(sizeof(ListNode));  
15     (!p)? exit(0):p->next=NULL;}  
16  
17 int main(){  
18     LinkNode *p = initList(); //正确  
19     //以下错误  
20     LinkNode *ptr1; initList1(ptr1);}
```

基本方法

- 全局变量：函数外定义变量；
- 定义返回值，例如 “int f(){...; return x;}"。可以返回一个结构体，带回多个返回值，但是有内存复制操作，低效；
- 利用函数（输入）参数带回结果，例如 “void f1(int *x){...}"