# 目录

[目录 1](#_Toc15456664)

[第10章 C语言-链表 2](#_Toc15456665)

[10.1 链表概述 2](#_Toc15456666)

[10.2 静态链表 2](#_Toc15456667)

[10.2.1 单向静态链表 2](#_Toc15456668)

[10.3 动态链表 2](#_Toc15456669)

[10.3.1 何谓动态链表 2](#_Toc15456670)

[10.3.2 申请内存空间 2](#_Toc15456671)

[10.3.3 释放内存空间 2](#_Toc15456672)

[10.3.4 建立单向动态链表 2](#_Toc15456673)

[10.3.5 遍历单向动态链表 2](#_Toc15456674)

[10.3.6 单向链表查询节点 2](#_Toc15456675)

[10.3.7 单向链表插入节点 2](#_Toc15456676)

[10.3.8 单向链表删除节点 2](#_Toc15456677)

[10.3.9 建立双向动态链表 2](#_Toc15456678)

[10.3.10 遍历双向动态链表 2](#_Toc15456679)

# C语言-链表

## 链表概述

数组：先确定大小，才能使用，换句话说，数组空间大小要申请好，才能取去使用它

int a[50];//定义了元素个数为50，元素类型是int类型的数组空间 200字节

存10个数据：空间浪费

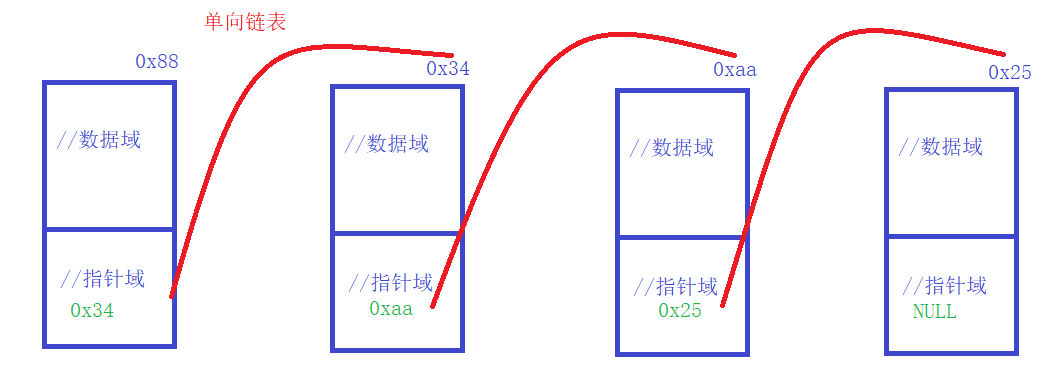
存60个数据：空间不足

链表：由多个结点组成的一种链状结构(结点 🡪 空间)

结点：由两部组成：数据域和指针域

数据域：存储数据

指针域：存储下一个结点的地址



1. 按存储方式
2. 静态链表：链表的长度是固定，地址是连续 🡪 结构体数组

特征：对一个数组元素进行访问，通过数组名[下标]就可以访问到该元素，只操作一次即可

对删除和插入操作，首先要找到删除和插入的位置，在进行移动元素

1. 动态链表：链表的长度不固定，地址也是不连续(指针)

特征：对一个结点进行访问，不能通过结点名字进行访问(结点没有名字)

对删除和插入操作，首先要找到删除和插入的位置。交换指针指向即可

1. 链表的结构(单向链表)
2. 头结点：链表的头，无论对链表做哪种操作，必须要保证有指针指向表头，表头的数据域一般不存数据，指针域存下一个结点地址
3. 结点：中间结点；数据域存储数据，指针域存储下一个结点地址
4. 尾结点：链表的尾，数据域存储数据，指针域为NULL
5. 链表的方向
6. 单向链表：上一个结点指针域存储下一个结点地址，表头数据域不存数据，表尾指针域为NULL
7. 单向循环链表：上一个结点指针域存储下一个结点地址，表尾指针域指向表头地址
8. 双向链表：一个结点有两个指针域，一个指针域指向上一个结点地址，另一个指针域指向下一个结点地址，表头的指向前一个结点的指针域为NULL，表尾指向下一个结点的指针域为NULL
9. 双向循环链表：一个结点有两个指针域，一个指针域指向上一个结点地址，另一个指针域指向下一个结点地址，表头的指向上一个结点的指针域指向表尾，表尾指向下一个结点的指针域指向表头
10. 结点的类型

结构体类型

结构体类型：结点里面可以存数据也可以存地址 🡪 数组不能实现

struct 结构体名

{

//数据域：存储数据

//指针域：存储下一个结点地址 指针变量

};

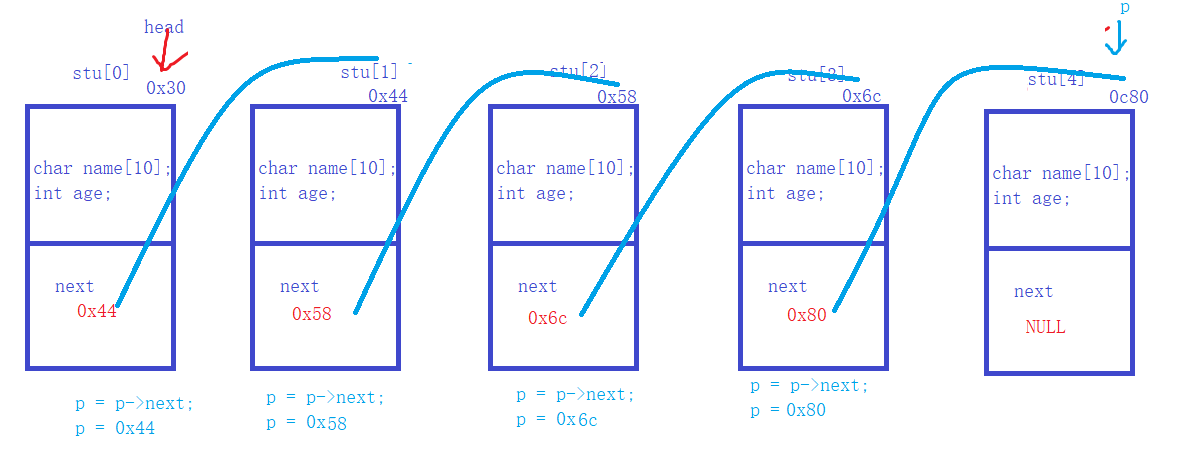
## 静态链表

链表的长度固定，地址连续 结构体数组

有头结点和尾结点的概念

头结点特征：数据域不存数据，指针域指向下一个结点地址

尾结点特征：数据域存储数据，指针域为NULL



|  |
| --- |
| #include <stdio.h>  //构造出结点类型  typedef struct NODE  {  //数据域  char name[10];  int age;  //指针域  struct NODE \*next;//结构体指针变量  }STU;  int main(void)  {  STU stu[5] = {  {"张三", 15},  {"刘志雄", 18},  {"李四", 18},  {"张五", 13},  {"张六", 19}  };  STU \*head = NULL;  STU \*p = NULL;  head = &stu[0];//head指向表头  stu[0].next = &stu[1];  stu[1].next = &stu[2];  stu[2].next = &stu[3];  stu[3].next = &stu[4];  stu[4].next = NULL;//表尾  p = head;  while(p->next != NULL)  {  p = p->next;  printf("%s %d\r\n", p->name, p->age);  }  return 0;  } |

## 动态链表

### 何谓动态链表

链表的长度不固定，地址不连续

之前学过的数据类型，不能实现临时改变空间大小

可以用内存管理函数，实现空间大小的切换

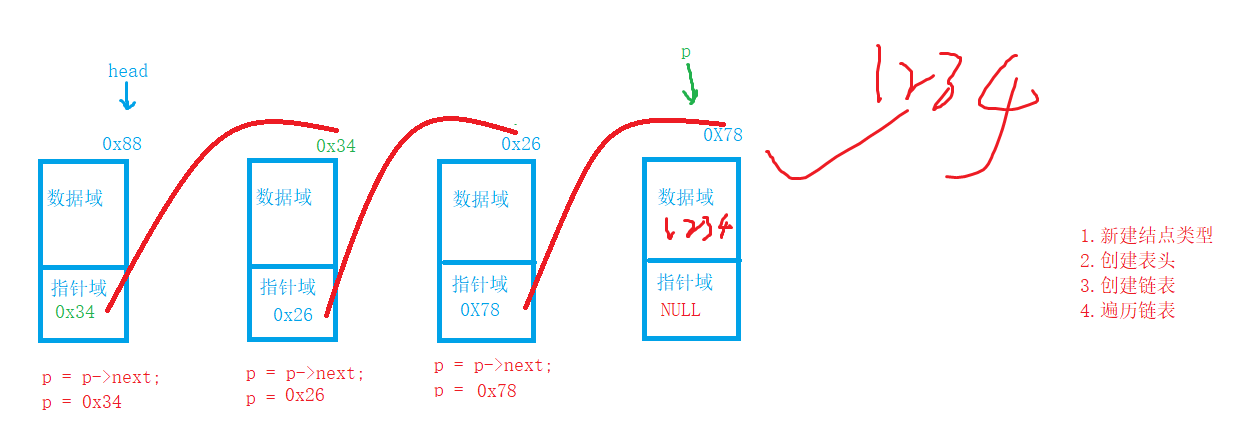
结点：可以通过内存管理函数来申请

需要结点：申请结点 malloc calloc

删除结点：释放结点 free

合理利用空间的资源

### 建立单向动态链表



1. 新建结点类型

|  |
| --- |
| //构造出结点类型  typedef struct NODE  {  //数据域  char name[10];  int number;  int math;  int english;  int chinese;  //指针域  struct NODE \*next;//结构体指针变量  }STU; |

1. 创建表头

|  |
| --- |
| /\*  函数功能：创建表头  函数形参：无  返回值：表头地址  备注：无  \*/  STU \*Link\_Create\_Head(void)  {  STU \*head = NULL;  head = (STU \*)malloc(sizeof(STU));//创建了一个头结点空间  if(head == NULL)  {  printf("头结点申请失败\r\n");  return NULL;  }  head->next = NULL;//保证尾结点的指针域为NULL  return head;//返回表头地址  } |

1. 创建链表

|  |
| --- |
| /\*  函数功能：创建链表  函数形参：表头地址  返回值：表头地址  备注：无  \*/  STU \*Link\_Ctreate\_Data(STU \*head)  {  STU \*p = head; //保存表头地址  STU \*f = NULL; //保存新建结点的首地址  char ch = 0;  while(1)  {  f = (STU \*)malloc(sizeof(STU));  //数据域  printf("请输入学生的姓名：");  scanf("%s", f->name);  printf("请输入学生的学号：");  scanf("%d", &f->number);  printf("请输入学生的数学成绩：");  scanf("%d", &f->math);  printf("请输入学生的英语成绩：");  scanf("%d", &f->english);  printf("请输入学生的语文成绩：");  scanf("%d", &f->chinese);  //指针域  p->next = f;  f->next = NULL;  p = f;  getchar();  printf("是否继续录入信息y/n:");  scanf("%c", &ch);  if(ch == 'n' || ch == 'N')  {  break;  }  }  printf("链表创建完成\r\n");  return head;  } |

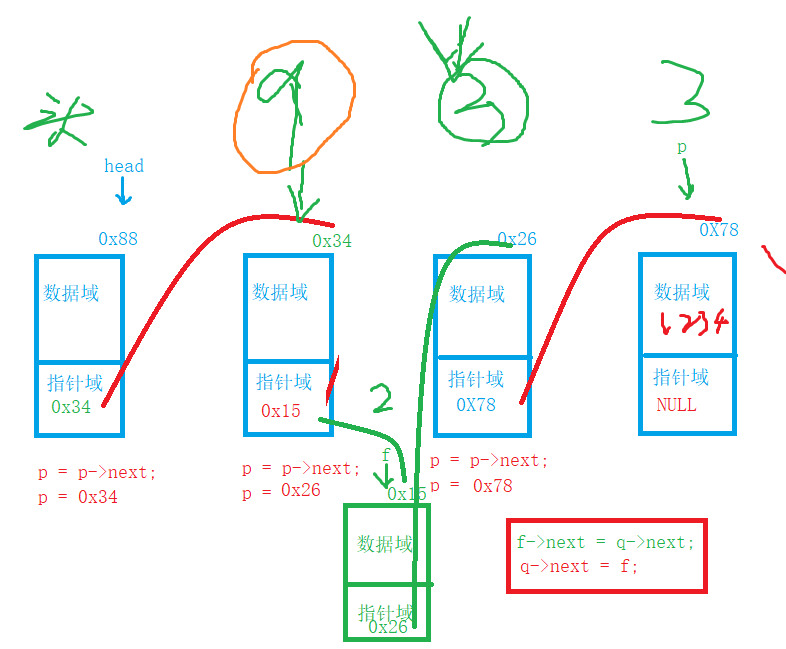
### 遍历单向动态链表

|  |
| --- |
| /\*  函数功能：输出链表信息(遍历链表)  函数形参：表头地址  返回值：无  备注：无  \*/  void Link\_Output\_Data(STU \*head)  {  STU \*p = head;  if(head == NULL)  {  printf("没有表头地址\r\n");  return ;  }  printf("姓名 学号 数学 英语 语文\r\n");  while(p->next != NULL)  {  p = p->next;  printf("%s %d %4d %4d %4d\r\n", p->name, p->number, p->math, p->english, p->chinese);  }  } |

### 单向链表查询节点

|  |
| --- |
| /\*  函数功能：按照学号查找，输出结点信息  函数形参：表头地址 学号  返回值：表头的地址  备注：无  \*/  STU \*Link\_Find\_Data(STU \*head, int xh)//3  {  STU \*p = head;  int flag = 0;    while(p->next != NULL)  {  p = p->next;  if(p->number == xh)  {  printf("%s %d %4d %4d %4d\r\n", p->name, p->number, p->math, p->english, p->chinese);  flag = 1;  break;  }  }    if(flag == 0)  {  printf("学号不存在!!\r\n");  }  return head;  } |

### 单向链表插入节点



|  |
| --- |
| /\*  函数功能：按位置插入结点  函数形参：表头地址 插入的位置  返回值：表头的地址  备注：无  \*/  STU \*Link\_Inser\_Node(STU \*head, int wz)  {  STU \*q = NULL;//保存要插入的前一个结点的地址  STU \*f = NULL;//用来保存插入结点的地址  q = Link\_Node\_Adrress(head, wz);//得到插入位置的前一个结点地址  if(q == NULL)  {  printf("位置有误!\r\n");  return NULL;  }  f = (STU \*)malloc(sizeof(STU));  //数据域  printf("请输入学生的姓名：");  scanf("%s", f->name);  printf("请输入学生的学号：");  scanf("%d", &f->number);  printf("请输入学生的数学成绩：");  scanf("%d", &f->math);  printf("请输入学生的英语成绩：");  scanf("%d", &f->english);  printf("请输入学生的语文成绩：");  scanf("%d", &f->chinese);  //指针域  f->next = q->next;  q->next = f;  printf("插入成功！！\r\n");  return head;  } |

### 单向链表删除节点

|  |
| --- |
| /\*  函数功能：按位置删除结点  函数形参：表头地址 删除的位置  返回值：表头的地址  备注：无  \*/  STU \*Link\_Delete\_Node(STU \*head, int wz)  {  STU \*q = NULL;//保存删除前一个结点地址  STU \*f = NULL; //保存当前要删除结点的地址  q = Link\_Node\_Adrress(head, wz);//得到要删除的前一个结点地址  f = q->next;//得到当前要删除的结点地址  q->next = f->next;  free(f);//释放删除的空间  f = NULL;    printf("删除成功!!\r\n");  return head;  } |

### 建立双向动态链表

双向链表：一个结点有两个指针域，一个指针域指向上一个结点地址，另一个指针域指向下一个结点地址，表头的指向前一个结点的指针域为NULL，表尾指向下一个结点的指针域为NULL

1. 创建结点类型

|  |
| --- |
| struct NODE  {  //数据域  char name[10];  int age;  //指针域  struct NODE \*q;//指向前一个结点  struct NODE \*next;//指向后一个结点  } |

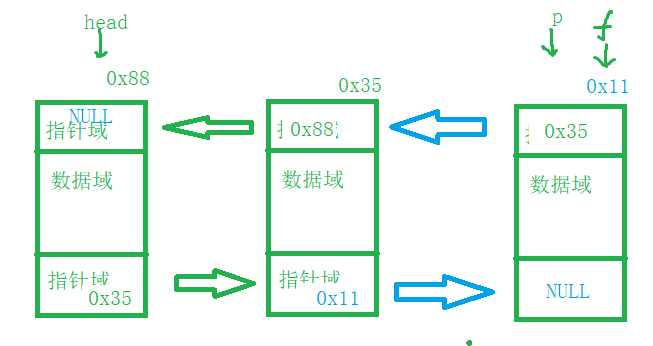
1. 创建表头

|  |
| --- |
| STU \*head = (STU \*)malloc(sizeof(STU));  head->q = NULL;  head->next = NULL; |

1. 创建链表

|  |
| --- |
| for(i = 0; i < n; i++)  {  f = (STU \*)malloc(sizeof(STU));  //数据域  printf("请输入姓名和年龄:");  scanf("%s %d", f->name, &f->age);  //指针域  p->next = f;  f->q = p;  f->next = NULL;  p = f;  } |

### 遍历双向动态链表



|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  typedef struct NODE  {  //数据域  char name[10];  int age;  //指针域  struct NODE \*q;//指向前一个结点  struct NODE \*next;//指向后一个结点  }STU;  int main(void)  {  STU \*p = NULL;  STU \*f = NULL;  int n = 0, i;  STU \*head = (STU \*)malloc(sizeof(STU));  head->q = NULL;  head->next = NULL;  p = head;    printf("请输入要录入学生的个数：");  scanf("%d", &n);  for(i = 0; i < n; i++)  {  f = (STU \*)malloc(sizeof(STU));  //数据域  printf("请输入姓名和年龄:");  scanf("%s %d", f->name, &f->age);  //指针域  p->next = f;  f->q = p;  f->next = NULL;  p = f;  }  //方法1  /\* p = head;  while(p->next != NULL)  {  p = p->next;  printf("%s %d\r\n", p->name, p->age);  }\*/  //方法2  while(p->q != NULL)  {  printf("%s %d\r\n", p->name, p->age);  p = p->q;  }  return 0;  } |