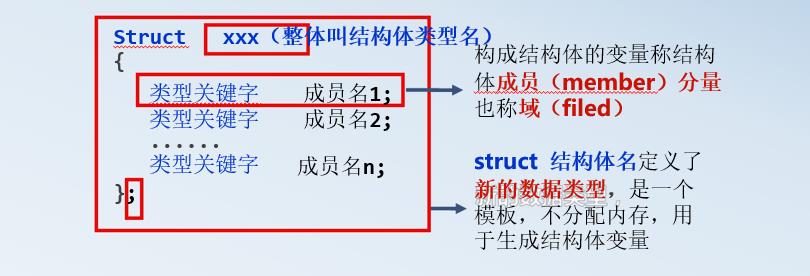
# 结构体

**结构体是一种自定义数据类型**，结构体**变量代表的结构体类型的数据对象作为一个整体存储在内存中。**

**结构体类型定义的基本语法形式为：**



#include <stdio.h>

#include <stdlib.h>

typedef struct student

{

int no;

char name[20];

float score;

}STU;

/\*输出成绩最高的人的学号\*/

int f(STU \*x,const int len)

{

int i=0;

int no;

float s=x[0].score;

while (i<len)

{

if (x[i].score>s)

{

no=i;

s=x[i].score;

}

i++;

}

return x[no].no;

}

int main(void)

{

STU stus[5]={{11,"zhangsan",80.0},{12,"lisi",59.5},{13,"wangwu",94.5},{14,"zhaoliu",98.5},{15,"dingqi",86.0}};

int s;

s=f(stus,5);

printf("%d",s);

system("pause");

return 0;

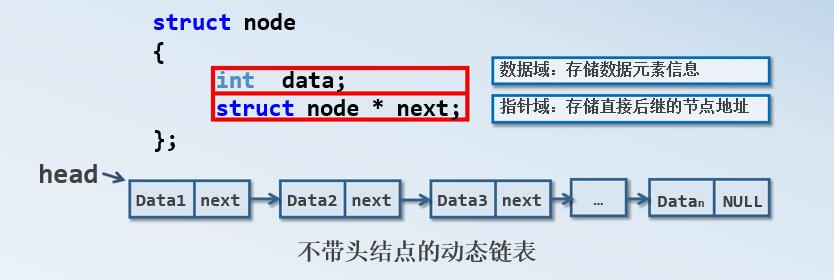
}

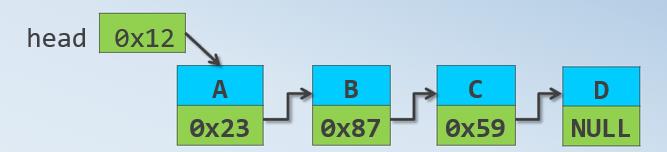
# 结构体与动态链表

动态数据结构-----**链表**

## 动态链表的构成：

动态链表有一个或者多个结点构成，每个节点都是一个结构体对象。每个结点有数据域和指针域(关系)





**注意：**① 链表有一个**头指针**，存放第一个结点的地址

② 每一个**结点**由数据域和指针域构成

③ 最后一个结点指针域为NULL，称为“**表尾**”

④ 结点通常就是一个结构体变量

**链表特点：**① 链表中各元素可以不是连续存放的.

② 要找某结点必须先找到上个结点因此必须提供头指针.

③ 创建结点就是创建结构体和利用指针做成员.

## 动态链表

动态链表的结点是临时生成的。

可以根据需要添加或者删除元素。

程序员手工分配内存与释放内存（堆区）。

* **Malloc**
* **Calloc**
* **Realloc**
* **free**

## 链表程序

### 创建一个链表并对它进行遍历

方法一：头查法

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

typedef struct Node

{

int x;

struct Node \*y;

}Node;

/\*创建一个链表\*/

Node \* f(int n)

{

Node \*p;

int i=0;

Node \*h=NULL;

while (i<n)

{

p=(Node \*)malloc(sizeof(Node));

printf("请输入第%d个元素",n-i);

scanf("%d",&(p->x));

p->y=h;

h=p;

i++;

}

return h;

}

/\*实现对链表的遍历\*/

void visit(Node \* l)

{

while (l!=NULL)

{

printf("%d\t",l->x);

l=l->y;

}

}

/\*检验\*/

int main(void)

{

Node \*h;

h=f(5);

visit(h);

system("pause");

return 0;

}

**方法二：尾查法（不会）**

### 求链表结点的个数

int number(Node \*l)

{

int i=0;

while (l!=NULL)

{

i++;**//若是求和则改为i=i+l->x;如果是正数之和,则需要进行判断。**

l=l->y;

}

return i;

}

### 请返回倒数第k个元素

int ff(Node \*l,int k,int n)

{

int i=0;

while (l!=NULL)

{

i++;

if (i==n-k+1)

return l->x;

l=l->y;

}

}

### 链表的查找和更新

问题描述：在链表中找到a元素将a元素找到后变换为b元素，并将更新后链表输出。

int ff(Node \*l,int a,int b)

{

while ((l!=NULL)&&(l->x!=a))

{

l=l->y;

}

if (l!=NULL)

{

l->x=b;

return 1;

}

return 0;

}

### 链表的逆置

typedef struct Node

{

int x;

struct Node \*y;

}Node;

Node \* inverse(Node \*l)

{

Node \*ph=NULL;

while (l!=NULL)

{

Node \*q=l;

l=l->y;

q->y=ph;

ph=q;

}

return ph;

}

### 链表的插入

**问题一：在第k个结点的前面插入一个结点，数据域是x**

Node \*ff(Node \*l,int k,int x)

{

Node \*h=l;

Node \*p;

int i=1;

if (k==1)

{

p=(Node \*)malloc(sizeof(Node));

p->x=x;

p->y=l;

l=p;

return l;

}

else

{

while (l!=NULL&&i<k-1)

{

l=l->y;

i++;

}

if (l!=NULL)

{

p=(Node \*)malloc(sizeof(Node));

p->x=x;

p->y=l->y;

l->y=p;

return h;

}

else

printf("不存在第k个结点！");

}

}

**问题二：在链表中在元素x之后插入一个结点元素值为y。**

Node \*ff(Node \*l,int x,int y)

{

Node \*h=l;

Node \*p;

while (l!=NULL&&l->x!=x)

{

l=l->y;

}

if (l!=NULL)

{

p=(Node \*)malloc(sizeof(Node));

p->x=y;

p->y=l->y;

l->y=p;

return h;

}

}

### 链表的删除

**问题一：删除第k个结点。**

Node \*ff(Node \*l,int k)

{

Node \*h=l;

Node \*p;

int i=1;

if (k==1)

{

p=l->y;

free(l);

return p;

}

else

{

while (l!=NULL&&i<k-1)

{

l=l->y;

i++;

}

if (l!=NULL)

{

p=l->y;

l->y=p->y;

free(p);

return h;

}

else

printf("不存在第k个结点！");

}

}

**问题二：在链表中删除元素x。**

Node \* f(int n)

{

Node \*p;

int i=0;

Node \*h=NULL;

while (i<n)

{

p=(Node \*)malloc(sizeof(Node));

printf("请输入第%d个元素",n-i);

scanf("%d",&(p->x));

p->y=h;

h=p;

i++;

}

return h;

}

Node \*ff(Node \*l,int x)

{

Node \*h=l;

Node \*p;

int k;

int i=0;

while (l!=NULL)

{

i++;

if (l->x==x)

k=i;

l=l->y;

}

if(k==1)

{

p=l->y;

free(l);

return p;

}

else

{

int j=1;

while (l!=NULL&&j<k-1)

{

l=l->y;

j++;

}

if (l!=NULL)

{

p=l->y;

l->y=p->y;

free(p);

return h;

}

else

printf("不存在第k个结点！");

}

}