指针

地址指向变量单元

eg：科大在玉泉路19号=在玉泉路19号可以找到科大

定义:

```
Type* pointer
```

eg:

```
void swap1(int *p,int *q){
    int temp;
    temp=*p1;
    *p1=*p2;
    *p2=temp;
}//right
void swap2(int *p,int *q){
    int *temp;
    *temp=*p1;
    *p1=*p2;
    *p2=temp;
}//wrong
```

数组VS指针

```
p=arr;
*(p+1)==*(a+1)==a[1]==p[1]
```

函数传参

```
void swap1(int a,int b){
    int tmp=a;
    a=b;
    b=tmp;
}//wrong
void swap2(int *a,int *b){
    int tmp=*a;
    *a=*b;
    *b=tmp;
}//right
void swap3(int *a,int *b){
    int *tmp=a;
    a=b;
    b=tmp;
}//wrong
```

多维数组

```
int arr[3][4];
int *p[3];//array of pointer
int (*q)[4];//pointer of array
q=arr;
p[0]=q[0];
arr[2][1]==*(*(q+2)+1)==q[2][1];
arr[2][1]==*(p[0]+2*4+1);
```

注意指针基类型

多级指针

```
int *p;
int **q;
```

指针数组

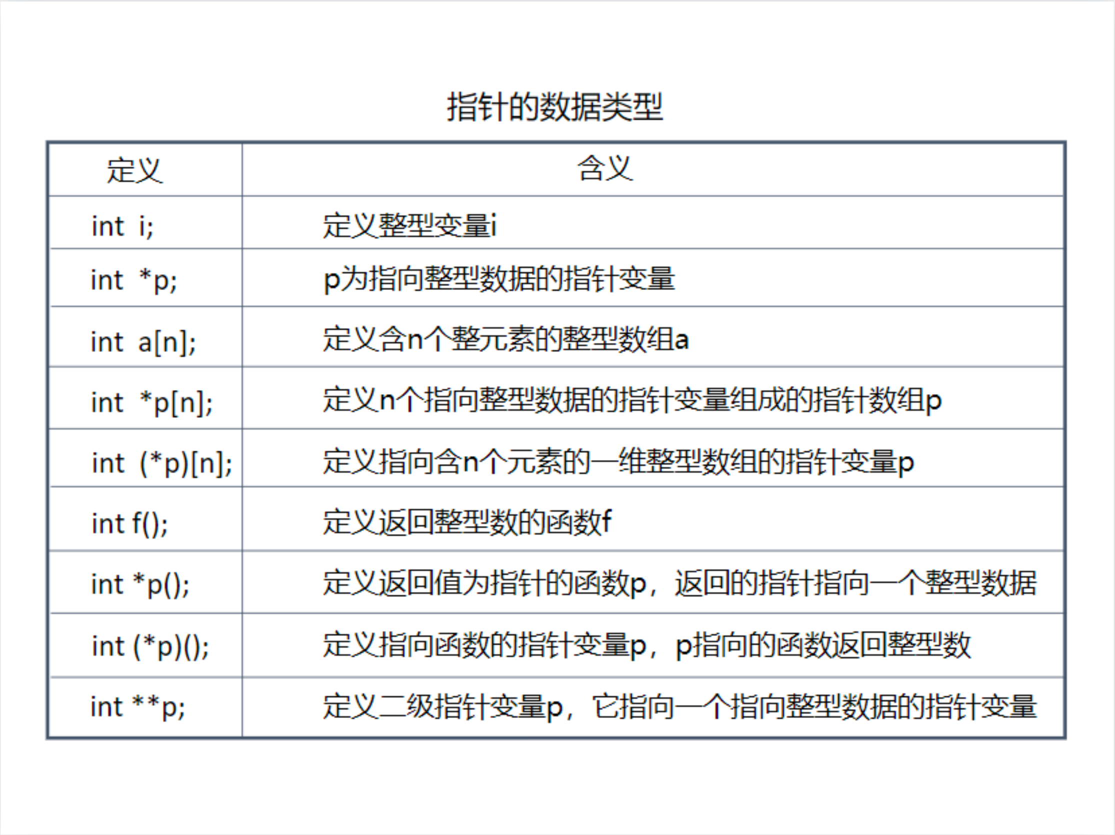
```
char *p[2]={"hallo","world"}
```

指针函数

```
int *f(int a,int b){
    return int *p;
}
```

动态分配内存

```
maolloc();
free();
```



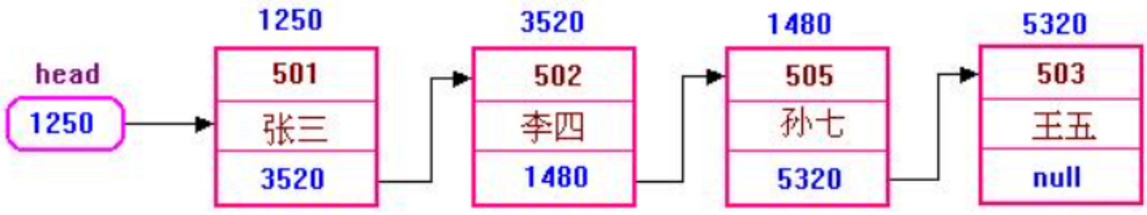
指针的数据类型

定义	含义
int i;	定义整型变量i
int *p;	p为指向整型数据的指针变量
int a[n];	定义含n个整元素的整型数组a
int *p[n];	定义n个指向整型数据的指针变量组成的指针数组p
int (*p)[n];	定义指向含n个元素的一维整型数组的指针变量p
int f();	定义返回整型数的函数f
int *p();	定义返回值为指针的函数p，返回的指针指向一个整型数据
int (*p)();	定义指向函数的指针变量p，p指向的函数返回整型数
int **p;	定义二级指针变量p，它指向一个指向整型数据的指针变量

结构体

```
struct name{
    Type1 member1;
    Type2 member2;
}name1,name2;
```

链表



```
struct student{
    int num;
    char name[6];
    struct student *next;
};
```

```
p=p->next;
p->next=q;
q->next=NULL;
```

建表

```
typedef int ElementType;

typedef struct Node {
    ElementType data;
    struct Node *Next;
} *List;
```

```
/*
    建立链表 - 头插法 - 没有头节点

              +-------+   +-------+    +-------+   +-------+
              | head | -> |node_1| ->  |node_2| -> |node_3| -> NULL
       ^      +-------+   +-------+    +-------+   +-------+
       |
    +-------+
    |  p   |
    +-------+

*/
List HeadCreate(void)
{
    ElementType x; // 保存 Node 中的 data 数据
    List p;
    List head;
    head = NULL;
    scanf("%d", &x);

    while (x != -1) {
        p = (List)malloc(sizeof(struct Node));
        p->data = x;
        if (head == NULL) { // 若第一次创建节点，则将该点设置为头节点
            head = p;
            p->Next = NULL;
        } else { // 若不是第一次创建节点，则直接将新节点接到链表头
            p->Next = head;
            head = p;
        }
        scanf("%d", &x);
    }
    return head;
}


/*
    创建链表 - 尾插法 - 没有头节点

    +-------+   +-------+   +-------+
    |node_1| -> |node_2| -> |node_3|    ____
    +-------+   +-------+   +-------+      |
                                           v
                            +-------+   +-------+
                            | rear | -> |   p  |
                            +-------+   +-------+

*/
List TailCreate(void)
{
    ElementType  x;
    List p;
    List head;
    List rear;
    head = NULL;
    rear = NULL;
    scanf(%d, &x);

    while (x != -1) {
        p = (List)malloc(sizeof(struct Node));
        p->data = x;
        if (head == NULL) { // 创建链表的第一个节点
            head = p;
            rear = p;
            p->Next = NULL;
        } else {
            rear->Next = p;
            rear = p;
        }
        scanf("%d", &x);
    }
    rear->Next = NULL; // 链表建立结束后将最后一个节点指向 NULL（尾插法中不要遗漏）
    return head;
}
```

查找

```
int search(Node *pNode,ElementType x){
    int pos;
    if (pNode == NULL){
        printf("%s函数执行，链表为空，查找x=%d失败\n",__FUNCTION__,x);
        return -1;
    }
    Node *pMove=pNode;
    while(pMove != NULL){
        if(pMove->data == x){
            return pos;
        }
        pMove = pMove->next;
        pos++;
    }
     if (pMove == NULL) {
        printf("%s函数执行，不存在x=%d，查找数据失败\n",__FUNCTION__,x);
        retu;
    }
}
```

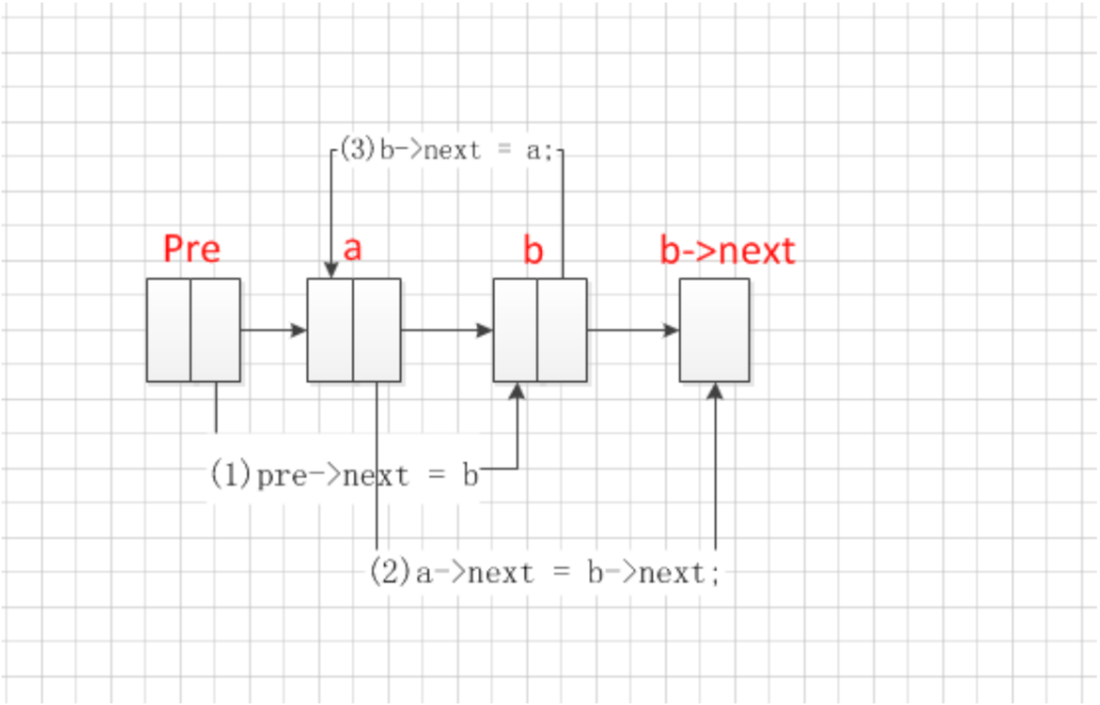
删除

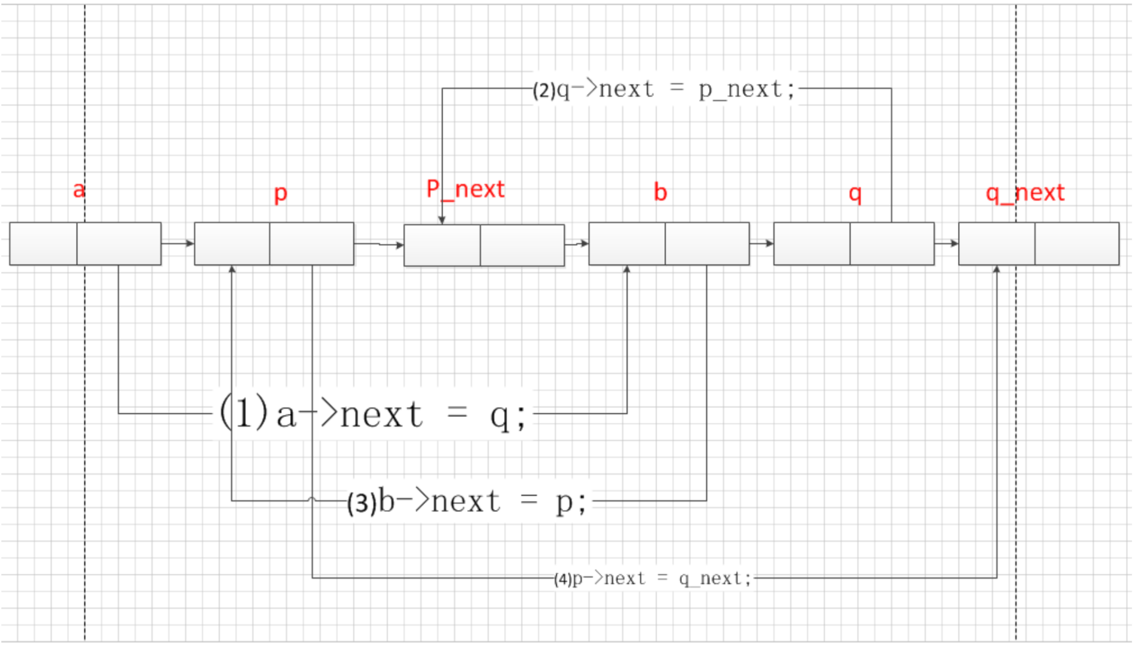
```
Node *delet(Node *pNode,ElementType x) {
    //一前一后两个指针，pMovePre是pMove的前一个节点
    Node *pMovePre;
    Node *pMove;

    if (pNode == NULL) {
        printf("%s函数执行，链表为空，删除x=%d失败\n",__FUNCTION__,x);
        return NULL;
    }

    pMovePre = pNode;
    pMove = pMovePre->next;

    //单独考虑第一个节点
    if (pMovePre->data == x) {
        pNode = pMove;
        free(pMovePre);
        return pNode;
    }

    while (pMove != NULL) {
        if (pMove->data == x) {
            //找到该节点的前一个节点
            pMovePre->next = pMove->next;
            free(pMove);
            break;
        }
        //同步前进
        pMove = pMove->next;
        pMovePre = pMovePre->next;
    }

    if (pMove == NULL) {
        printf("%s函数执行，不存在x=%d，删除数据失败\n",__FUNCTION__,x);
        return pNode;
    }

}


//删除pos位置的节点
Node *deletePosElement(Node *pNode,int pos){
    //需要一个头结点来维护
    Node *pHead;

    Node *pMove;
    int i = 1;
    if (pos <= 0 || pos > sizeList(pNode)) {
        printf("%s函数执行，输入pos值非法，删除节点失败\n",__FUNCTION__);
        return NULL;
    }

    pHead = pNode;
    pMove = pNode;
    //单独考虑删除第一个节点
    if (pos == 1) {

        pMove = pMove->next;
        pNode = pMove;
        free(pHead);

        printf("%s函数执行，删除pos=1位置元素成功\n",__FUNCTION__);
        return pNode;
    }

    while (pMove != NULL) {
        if (i == pos - 1) {
            break;
        }
        i++;
        pMove = pMove->next;
    }

    free(pMove->next);
    pMove->next = pMove->next->next;

    printf("%s函数执行，删除pos=%d位置元素成功\n",__FUNCTION__,pos);

    return pNode;
}
```

插入

```
int insert(Node *pNode,ElementType pilot;ElementType x) {
    //一前一后两个指针，pMovePre是pMove的前一个节点
    if(!search(pNode,pilot)){
        return 0;
    }

    Node *pMovePre;
    Node *pMove;

    Node *pos=(Node*)malloc(sizeof(Node));
    pos->data=x;
    pos->next=NULL;

    pMovePre = pNode;
    pMove = pMovePre->next;



    while (pMove != NULL) {
        if (pMove->element == pilot) {
            //找到该节点的前一个节点
            pMovePre=pMove;
            pMove=pMove->next;
            pMovePre->next=pos;
            pos->next=pMove;
            return 1;
        }

        pMove = pMove->next;
    }
}
```

排序





```
void swap(Node* p_pre,Node* q_pre){
    //p is before q
    if(p_pre->next==q_pre){//frist graph
        Node* Pre = p_pre;
        Node* a = q_pre;
        Node* b = a->next;
        pre->next = b;
        a->next=b->next;
        b->next=a;
    }
    else{//second
        Node* a = p_pre;
        Node* p = p_pre->next;
        Node* p_next = p->next;
        Node* b = q_pre;
        Node* q = q_pre->next;
        Node*q_next = q->next;
        a->next = q;
        q->next = p_next;
        b->next = p;
        p->next =q_next;
    }
}
```

```
Link insert(Node* head){
    Node* p_pre,p,q_pre,q,max_pre,max;
   //first loop
    max=head;
    q_pre=head;
    q=q_pre->next;
    while(q!=NULL){
        if(max->data<q->data){
           max = q;
           max_pre = q_pre;
        }
        q_pre=q;
        q=q->next;
    }
    if(max!=head){
        max_pre->next = head;
        Node* max_next;
        max_next = max->next;
        max->next = head->next;
        head->next = max_next;
        head = max;
    }
    p_pre = head;
    p=p_pre->next;
    while(p!=NULL){
        max=p;
        q_pre=p;
        q=q_pre->next;
        while(q!=NuLL){
            if(max->data<q->data){
                max = q;
                max_pre = q_pre;
            }
            q_pre=q;
            q=q->next;
        }
        if(max!=p){
            swap(p_pre,max_p)
        }
        p=p-next;
```

```
            p_pre=p;
        }
}
```

考

循环　数组　结构（体）　函数　指针　链表

读递归程序　排序、查找等简单的算法应用