第九讲 线性群体

1. 不定长动态数组array

Template <class t>

Class array{

T\* list;

Int size;

Public:

Array()

}

重载指针运算符：将array类型转换为t类型指针

Return list //因为list正好就是t类型的指针

//“断言”函数assert( …… )

作用： 如果满足括号内条件，则继续进行。

如果不满足条件，则直接结束当前函数的调用。

Void array<t>::resize(int sz){ //重置数组大小函数

Assert(sz>=0);

If(sz==size) return; //新的大小与原来一样，则不操作

先申请sz大小的内存

再比较新旧空间大小，复制数量为较小者的数组内容

……

}

1. 链表
2. 单链表：每个节点包含一个数据域和一个指向后一个节点的指针域
   1. Insert after：

在当前节点后插入：

template <class T>

void Node<T>::insertAfter(Node<T> \*p) {

   //把当前节点的后继作为新节点的后继

   p->next = this->next;

    //新节点作为当前节点的后继

this-> next = p;

}

* 1. Delete after:

在当前节点后删除：

template <class T>

Node<T> \*Node<T>::deleteAfter(void) {

if (this->next == 0) return 0;

Node<T> \*tempPtr = this-> next;

this-> next = tempPtr->next;

return tempPtr;

}

1. 双链表：每个节点有两个指针prior和next，分别指向前驱节点和后继节点

表头节点的前驱与表尾节点的后继为null

//双向链表在构造时就将表头的的前驱设置为了null

1. 在中间插入新节点
   1. 当前节点a1的后继a2作为新节点的后继
   2. 新节点作为当前节点的后继节点a2的前驱
   3. 当前节点a1作为新节点的前驱
   4. 新节点作为当前节点a1的后继
2. 在表头插入新节点
   1. 当前头节点作为新节点的后继；
   2. 新节点作为头节点的前驱；
   3. 新节点的前驱设置为null；
   4. 新节点作为头节点。
3. 在中间删除旧节点（在前面增加假的头节点以方便操作）
   1. 待删除节点（\*p的后继）的地址和data暂存，用于空间释放和数据返回
   2. 当前节点作为待删除节点的后继节点的前驱
   3. 待删除节点的后继节点作为当前节点的后继
   4. 释放待删除节点的空间
   5. 返回删除节点的data
4. 在表头删除旧节点
   1. 当前头节点的后继作为表头；
   2. 新表头的前驱设置为null。
5. 链表的遍历

Node \*p=m\_head; p!=NULL; p=p->m\_next

//从头结点到尾结点，用next来遍历，尾结点的后继为null