20220401-C++&数据结构

1.过程描述

1.1 C++ Prime

初始化列表 C++ 复制代码

```
1.使用初始化列表
DemoClass::DemoClass(string &fn,string &ln):firstname(fn),lastname(ln){}

2.不使用初始化列表
DemoClass::DemoClass(string &fn,string &ln)
{
    firstname=fn;
    lastname=ln;
}

区别在于，不使用初始化列表将调用string的默认构造函数，再调用string的函数运算符将firstname设置为fn，而初始化列表可以减少一个步骤，直接使用string的复制构造函数将firstname设置为fn
```

继承 C++ 复制代码

```
1.创建派生类对象时，程序首先调用基类构造函数（初始化继承的数据成员），然后再调用派生类构造函数（初始化新增的数据成员）。派生类的构造函数总是调用一个基类构造函数，可以通过初始化列表知名要使用的基类构造函数，否则将使用默认的。派生类对象过期时，程序将首先调用派生类析构函数，然后调用基类析构函数

2.基类指针可以在不进行显式类型转换的情况下指向派生类对象，基类引用可以在不进行显式类型转换的情况下引用派生类对象（反过来则不行）：
Derived player(1000,"Hello");
BaseClass *pt=&player;
BaseClass &rt=player;
但是基类指针或引用只能调用基类方法，不能用来调用派生类的方法
```

1.2 数据结构

线性表的顺序表示和实现 C++ 复制代码

```
多项式的顺序存储结构
#define MAXSIZE 100
typedef struct
{
    float coef;
    int expn;
}Polynomial;

typedef structs
{
    Polynomial * elem;
    int length;
}SqList;

1.初始化
Status InitList(SqList &L)
{
    L.elem=new ElemType[MAXSIZE];
    if(!L.elem)
        exit(OVERFLOW);
    L.length=0;
    return OK;
}

2.取值
Status GetElem(SqList L,int i,ElemType &e)
{
    if(i<1||i>L.length)
        return ERROR;
    e=L.elem[i-1];
    return OK;
}

3.查找
int LocateElem(SqList L,ElemType e)
{
    for(i=0;i<L.length;i++)
        if(L.elem[i]==e)
            return i+1;
    return 0;
}

4.插入
Status ListInsert(SqList &L,int i,ElemType e)
{
```

```
46        if((i<1||(i>L.length+1))
47           return ERROR;
48        for(j=L.length-1;j>=i-1;j--)
49           L.elem[j+1]=L.elem[j];
50        L.elem[i-1]=e;
51        ++L.length;
52        return OK;
53    }
54
55    5.删除
56    Status ListDelete(SqList &L,int i)
57    {
58        if((i<1)||(i>
```

线性表的链式表示和实现 C++ 复制代码

```
单链表的存储结构
typedef struct
{
    ElemType data;
    struct LNode *next;
}LNode,*LinkList;

1.初始化
Status InitList(LinkList &L)
{
    L=new LNode;
    L->next=nullptr;
    return OK;

2.取值
Status GetElem(LinkList &L,int i,ElemType& e)
{
    p=L->next;
    j=1;
    while(p&&j<i) //逻辑与
    {
        p=p->next;
        ++j;
    }
    if(!p||j>1) //逻辑或
        return ERROR;
    e=p->data;
    return OK;
}

3.查找
LNode* LocateElem(LinkList L,Elemtype e)
{
    p=L->next;
    while(p!&&p->data!=e)
        p=p->next;
    return p;
}

4.插入
Status ListInsert(LinkList &L,int i,ElemType e)
{
    p=L;
    j=1;
    while(p&&(j<i-1))
```

```
        p=p->next;
    s=new LNode;
    s->data=e;
    s->next=p->next
    p->next=s;
    return OK;
}

5.删除
Status ListDelete(LinkList &L,int i)
{
    p=L;
    j=0;
    while(p->next&&(j<i-1))
        p=p->next;
        ++j;
    if(!(p->next)||(j>i-1))
       return ERROR;
    q=p->next;
    p->next=q->next;
    delete q;
    return OK;
}

6.前插法创建单链表
void CreateList_H(LinkList &L,int n)
{
    L=new LNode;
    L->next=nullptr;
    for(i=0;i<n;++i)
    {
        p=new LNode;
        cin>>p->data;
        p->next=L->next;
        L->next=p;
    }
}

7.后插法
void CreateList_R(LinkList &L,int n)
{
    L=new LNode;
    L->next=nullptr;
    r=L;
    for(i=0;i<n;++i)
    {
        p=new LNode;
        cin>>p->data;
```

```
            p->next=nullptr;
            r->next=p;
            r=p;
        }
    }
```

双向链表 C++ 复制代码

```
双向链表的存储结构
typedef struct DulNode
{
    ElemType data;
    struct DulNode *prior;
    struct DulNode *next;
}DulNode,*DulLinkList;
双向链表存在以下关系：
d->next->prior=d->prior->next

1.插入
Status ListInsert_Dul(DulLinkList &L,int i,ElemType e)
{
    if(!(p=GetElem_Dul(L,i))
       return ERROR;
     s=new DulNode;
     s->data=e;
     s->prior=p->prior;
     p->prior->next=s;
     s->next=p;
     p->prior=s;
     return OK;
}

2.删除
Status ListDelete_Dul(DulLinkList &L,int i)
{
    if(!(p=GetElem_Dul(L,i))
       return ERROR;
    p->prior->next=p->next;
    p->next->prior=p->prior;
    delete p;
    return OK;
}
```

有序表的合并 C++ 复制代码

```
1.问题（集合合并）：
A=(1,2,3,4);
B=(2,3,4,5,7,8);
则C=(1,2,3,4,5,7,8);

1)顺序有序表的合并（时间复杂度和空间复杂度均为O(m+n)）
void MergeList_Sq(SqList LA,SqList LB,SqList &LC)
{
    LC.length=lA.length+LB.length;
    LC.elem=new ElemType[LC.length];
    pc=LC.elem;//指向新表的第一个元素
    pa=LA.elem;//指向第一个元素
    pb=LB.elem;
    pa_last=LA.elem+LA.length-1;//指向最后一个元素
    pb_last=LB.elem+LB.length-1;
    while((pa<=pa_last)&&(pb<=pb_last))
    {
        if(*pa<=*pb)
            *pc++=*pa++;
        else
            *pc++=*pb++;
    }
    while(pa<=pa_last)//LB已经到达表尾，依次将LA的剩余元素插入LC最后
        *pc++=*pa++;
    while(pb<=pb-last)
        *pc++=*pb++;
}

2)链式有序表的合并（时间复杂度为O(m+n)，空间复杂度为O(1)）
void MergeList_L(LinkList &LA,LinkList &LB,LinkList &LC)
{
    pa=LA->next;
    pb=LB->next;
    LC=LA;
    pc=LC;
    while(pa&&pb)
    {
        if(pa->data<=pb->data)
        {
            pc->next=pa;
            pc=pa;
            pa=pa->next;
        }
        else
        {
```

```
            pc->next=pb;
            pc=pb;
            pb=pb->next;
        }
    }
    pc->next=pa?pa:pb;//将非空表的剩余段插入到pc所指节点之后
    delete LB;
}
```

在归并两个链表时，不需要新建新表的节点空间，只需重新整理节点之间的关系就行，因此空间复杂度为O(1)

稀疏多项式运算 C++ 复制代码

```
数据结构定义：
typedef struct PNode
{
    float ecof;
    int expn;
    struct PNode *next;
}PNode,*Polynomial;

1.多项式的创建(时间复杂度为O(n^2))
void CreatePolyn(Polynomial *P,int n)
{
    P=new PNode;
    P->next=nullptr;
    for(i=1;i<=n;++i)
    {
        s=new PNode;
        cin>>s->coef>>s->expn;
        pre=P;
        q=P->next;
        while(q&&q->expn<s->expn)
        {
            pre=q;
            q=q->next;
        }
        s->next=q;
        pre->next=s;
    }
}

2.多项式的相加
void AddPolyn(Polynomial &Pa,Polynomial &Pb)
{
    pl=Pa->next;
    p2=Pb->next;
    p3=pa;
    while(p1&&p2)
    {
        if(p1->expn==p2->expn)
        {
            sum=p1->coef+p2->coef;
            if(sum!=0)
            {
                p1->coef=sum;
                p3->next=p1;
                p3=p1;
```

```
                p1=p1->next;
                r=p2;
                p2=p2->next;
                delete r;
            }else
            {
                r=p1;
                p1=p1->next;
                delete r;
                r=p2;
                p2=p2->next;
                delete r;
            }
        }else if(p1->expn<p2->expn)
        {
            p3->next=p1;
            p3=p1;
            p1=p1->next;
        }else
        {
            p3->next=p2;
            p3=p2;
            p2=p2->next;
        }
    }
    p3->next=p1?p1:p2;
    delete Pb;
}

```

2.结果输出

今天主要看了一部分继承的内容以及数据结构的线性表部分，基本概念倒是不难掌握，主要还是代码能力比较弱，不太能一下子看明白作者的代码，导致看起来还挺费劲的。晚上本来尝试结合QT编一个简单的链表创建及添加程序，发现还是有很多觉得模棱两可的地方，明显感觉基础知识掌握不牢固。学习确实有它自身的规律，练习不够就不能指望一步登天。还是得脚踏实地。明天继续数据结构的队列跟栈部分。