

**《数据结构》**

**实验报告**

**实验（ 一 ）**

**年 级： 2023级**

**专 业： 计算机科学与技术**

**姓 名： 袁田**

**学 号： 2314022**

# 1 实验内容

实现以下内容：

1.从终端输入两个数列，分别存储在两个链表A和B中，并将两个链表的内容交替合并，合并成一个新的链表C，输出ABC三个链表。

2.对链表C，删除重复元素得到链表D并输出，对于链表中的重复元素要求保留第一次出现的元素。

3.输入一个整数k，将链表D按从左至右的顺序分成多个长度为k的区间，对于每个区间：

3.1 如果区间内节点数等于k则将该区间节点顺序反转

3.2 如果最后一个区间节点数小于k则保持原有顺序

按照上述要求输出结果链表E。

# 2 设计思想

# 该实验的设计思想如下：

# 1.使用类来构造链表，让链表有插入，删除，查找等功能。

# 2.合并形成一个新的链表：

通过使用链表的插入功能，由于设计链表时，插入函数的两个参数分别是插入位置和插入数据，可以使用链表的size来实现位置的不断的后移，而数据通过一个和头指针相同的指针进行遍历来获得。具体实现的过程中，由于两链表长度不一致，需要先确定较小长度，然后依次交替插入，再根据那一个链表更长分开讨论，插入剩下的数据。

1. 去重：

设计一个新的链表，使用查找功能（即确定某个数据是否在链表中的函数）遍历合并后的链表时确定某元素是否在新链表中存在，如果已经存在则放弃插入，如果不存在则插入到新链表中。

1. k个进行反转

使用stack这个数据结构，由于stack的特性(先放入的元素在stack的底部，后放入的元素在stack的头部可以用top读出来），首先确定按k个是否可以分为整数组，依次来分类讨论。

# 3 程序效果

程序的运行效果，输入及输出的相关要求和具体执行结果如下所示：

## 3.1 输入

2 4 8 1 5

H 5 C D E 4 F G

3

## 3.2 输出

2 4 8 1 5

H 5 C D E 4 F G

2 H 4 5 8 C 1 D 5 E 4 F G

2 H 4 5 8 C 1 D E F G

4 H 2 C 8 5 E D 1 F G

# 4 核心代码

**Part 1:链表的构造**

struct node

{

char data;

node\* next;

node(int val)

{

data=val;

next=NULL;

}

node()

{

next=NULL;

}

};

class Linklist

{

public:

int size;

node\* head;

Linklist();

~Linklist();

void Insert(int pos,char val);

void Delete(int pos);

void Print();

bool Find(char val);

};

Linklist::Linklist()

{

size=0;

head=new node;

}

Linklist::~Linklist()

{

if(head==NULL) return;

node\* tmp=head;

for(int i=0;i<size;i++)

{

head = tmp;

tmp = head->next;

delete head;

}

size=0;

}

void Linklist::Insert(int pos,char val)

{

if(pos<0||pos>size) return;

if(head==NULL) return;

node\* tmp=head;

for(int i=0;i<pos;i++)

{

tmp=tmp->next;

}

node \*insertnode=new node(val);

insertnode->next=tmp->next;

tmp->next=insertnode;

size++;

}

void Linklist::Delete(int pos)

{

if(pos<0||pos>=size) return;

if(head==NULL) return;

node\* tmp=head;

for(int i=0;i<pos;i++)

{

tmp=tmp->next;

}

node\* DeletePoint = tmp->next;

tmp->next = DeletePoint->next;

delete DeletePoint;

size--;

}

void Linklist::Print()

{

if(head==NULL) return;

node\* tmp=head->next;

for(int i=0;i<size;i++)

{

cout<<tmp->data<<" ";

tmp=tmp->next;

}

cout<<endl;

}

bool Linklist::Find(char val)

{

if(head==NULL) return false;

node\* tmp=head->next;

for(int i=0;i<size;i++)

{

if(tmp->data!=val)

{

tmp=tmp->next;

}

else

return true;

}

return false;

}

**Part 2:输入两个数列放入链表并交替放入第三个链表**

Linklist A,B;

string s,S;

getline(cin,s);

getline(cin,S);

for(int i = 0;i < s.size();i++)

{

if(s[i]==' ')continue;

A.Insert(A.size,s[i]);

n++;

}

时间复杂度：Θ(n²) 空间复杂度：O(n\*sizeof(node))

for(int i = 0;i < S.size();i++)

{

if(S[i]==' ')continue;

B.Insert(B.size,S[i]);

m++;

}

A.Print();

B.Print();

时间复杂度：Θ(m²) 空间复杂度：O(m\*sizeof(node))

//交替合并

Linklist C;

node\* tmp1=A.head->next;

node\* tmp2=B.head->next;

for(int i=0;i<min(n,m);i++)

{

C.Insert(C.size,tmp1->data);

C.Insert(C.size,tmp2->data);

tmp1=tmp1->next;

tmp2=tmp2->next;

}

if(n<m)

{

while(tmp2!=NULL)

{

C.Insert(C.size,tmp2->data);

tmp2=tmp2->next;

}

}

if(n>m)

{

while(tmp1!=NULL)

{

C.Insert(C.size,tmp1->data);

tmp1=tmp1->next;

}

}

1. Print();

时间复杂度：Θ(（n+m）²) 空间复杂度：O((n+m)\*sizeof(node))

**Part 3:去重**

Linklist D;

int len=n+m;

node\* tmp=C.head->next;

for(int i=0;i<len;i++)

{

if(!D.Find(tmp->data))

{

D.Insert(D.size,tmp->data);

}

tmp=tmp->next;

}

1. Print();

时间复杂度：如果去掉重复后C中的元素数目为X，则时间复杂度为：O(X²)

空间复杂度：如果去掉重复后C中的元素数目为X,则空间复杂度为O(X\*sizeof(node))

**Part 4:按k个反转**

cin>>k;

stack<char>p;

Linklist E;

if(D.size%k==0)

{

node\* tmp=D.head->next;

for(int i=0;i<D.size/k;i++)

{

for(int j=0;j<k;j++)

{

p.push(tmp->data);

tmp=tmp->next;

}

for(int q=0;q<k;q++)

{

E.Insert(E.size,p.top());

p.pop();

}

}

}

else if(D.size%k!=0)

{

node\* tmp=D.head->next;

for(int i=0;i<D.size/k;i++)

{

for(int j=0;j<k;j++)

{

p.push(tmp->data);

tmp=tmp->next;

}

for(int q=0;q<k;q++)

{

E.Insert(E.size,p.top());

p.pop();

}

}

while(tmp!=NULL)

{

E.Insert(E.size,tmp->data);

tmp=tmp->next;

}

}

1. Print();

如果去掉重复后C中的元素数目为X

时间复杂度：Θ(X) 空间复杂度：O(X\*sizeof(node))

# 5 总结

利用单向链表存储数据，我熟悉了单向链表，可以独立写出简单的链表功能。使用栈的数据结构完成数据的反转，由于stack独特的功能，先放入在底部，后放入在头部，有效的实现了数据的反转。

加深了对时间复杂度和空间复杂度的认识。