历年真题

2024年11月23日 21:11

//2012年真题，将一个链表分成奇次项子链表和偶次项子链表

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>

using namespace std;

typedef struct node
{
	int data;
	struct node* next;
}node;
bool initial(node * L)
{
	L = new node;
	L->next = NULL;
	return true;
}
int len(node * L)
{
	int len = 0;
	while (L!= NULL)
	{
		L = L->next;
		len = len + 1;
	}
	return len;
}
void display(node* L)
{
	cout << endl;
	while (L->next!= NULL)
	{
		cout << L->data << "\t";
		L = L->next;
	}
	cout << L->data;		//注意：要多输出最后一个数！！！
}
void insertladder(node* L, int a)
{
	node* p = L;
	while (p->next != NULL)p = p->next;
	node* pp = new node;
	p->next = pp;
	pp->data = a;
	pp->next = NULL;

}
node* inserthead(node* L, int a)
{
	node* p = L;
	node* pp = new node;
	pp->data = a;
	pp->next = NULL;
	return pp;
}
void insert(node* L)	//很好用！！直接输入便是，输入-1表示输入结束！！
{
	int a = 0;
	node* p = new node;
	node* pt = new node;
	pt = p = L;
	cin >> a;
	if (a != -1) p->data = a;

	while (1)
	{
		pt = p;
		cin >> a;
		if (a == -1) break;
		p = new node;
		pt->next = p;
		p->data = a;
		p->next = NULL;
	}
}


void divided_(node* L, node*& A, node*& B)//注意：这里应当传入指针的引用!!!!否则就是值传递！！ 将L链表分割成奇次项子链表A和偶次项子链表B
{
	int count = 0;
	node* pa = A;
	node* pb = B;
	pa->data = L->data;
	L = L->next;
	pb->data = L->data;
	L = L->next;
	while (L!= NULL)
	{
		if (count % 2 == 0)
		{
			node* paa = new node;
			pa->next = paa;		//pa->next
			paa->data = L->data;		//初始化paa
			paa->next = NULL;
			pa = paa;			//pa后移

		}
		else
		{
			node* pbb = new node;
			pb->next = pbb;
			pbb->data = L->data;
			pbb->next = NULL;
			pb = pbb;
		}
		count++;
```

简化步骤：不用new新节点!!

```
void merge__(node* A, node* B, node*& C)		//看哪个链表头小直接插在pc的后面。
{
	node* pa = A;
	node* pb = B;
	node* pc = C;
	if (pa->data <= pb->data)
	{
		pc->data = pa->data;
		pa = pa->next;
	}
	else
	{
		pc->data = pb->data;
		pb = pb->next;

	}
	while (pa != NULL && pb != NULL)
	{
		if (pa->data <= pb->data)
		{
			pc->next = pa;		//pa小就把pa插在pc后面
			pc = pc->next;
			pa = pa->next;
		}
		else
		{
			pc->next = pb;
			pc = pc->next;
			pb = pb->next;
		}

	}
	if (pa != NULL)	//谁有剩余就插在pc后面
	{
		pc->next = pa;
	}
```

```
                count++;
                L = L->next;
        }
}
int main()
{
        node* A = new node;
        node* B = new node;
        node* C = new node;
        insert(A);
        display(A);
        divided_(A,B,C);
        display(B);
        display(C);
        return 0;
}
```

```
        else if (pb != NULL)
        {
                pc->next = pb;
        }
}
```

//2013翻转链表，空间复杂度为一

```
void reverse(node*& L)
{
        node* prev = NULL;
        node* p = L;
        node* next;
        while (p != NULL)
        {
                next = p->next;         //保存p->next的地址
                p->next = prev;         //将p->prev的值赋给p->next
                prev = p;                       //保存prev的值为p
                p = next;                       //p指向下一个位置
        }
        L = prev;
}

int main()
{
        node* A = new node;

        insert(A);
        display(A);

        reverse(A);
        display(A);
        return 0;
}
```

2013翻转线性表，空间复杂度为1

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>

using namespace std;
typedef struct
{
        int* data;                      //注意类型是int*类型
        int len;
}sqlist;
void display(sqlist &L)         //始终要记得，不加&就是值传递，会造成浅拷贝！！！！！
{
        cout << endl;
        for (int i = 0; i < L.len; i++)
        {
                cout << L.data[i]<<'\t';
        }
        cout << "len=" << L.len;
}
void add(sqlist &L)
{
        int a;
        while (1)
        {
                cin >> a;
                if (a == -1)break;
                L.data[L.len++] = a;
        }

}
void reverse(sqlist& L)     //翻转很脑瘫啊，直接就是交换第len-i个和第i个数就可以了
{
        int temp = 0;
        int num = L.len / 2;
        for (int i = 0; i < num; i++)
        {
                temp = L.data[L.len - i-1];
                L.data[L.len-i-1] = L.data[i];
                L.data[i] = temp;

        }
}
int main()
{
        sqlist A;
        A.len = 0;
        A.data = new int[500];      //注意要进行初始化，不然要报错
        add(A);
        display(A);
        reverse(A);
        display(A);
        return 0;
}
```

//拓展：线性表删除第m个数，在第m个位置插入数

```
void add_m(sqlist& L, int m,int a)
{
        for (int i = L.len; i > m - 1; i--)
        {
                L.data[i] = L.data[i-1];
        }
        L.data[m] = a;
        L.len = L.len + 1;
}
void deleted(sqlist& L, int m)
{
        for (int i = m-1; i < L.len; i++)
        {
                L.data[i] = L.data[i + 1];
        }
        L.len = L.len - 1;
}
```

//2024归并：两个有序链表归并成一个更大的有序链表！！

```
void merge__(node* A, node* B, node*& C)          //看哪个链表头小直接插在pc的后面。
{
        node* pa = A;
        node* pb = B;
        node* pc = C;
        if (pa->data <= pb->data)
        {
                pc->data = pa->data;
                pa = pa->next;
        }
        else
        {
                pc->data = pb->data;
                pb = pb->next;

        }
        while (pa != NULL && pb != NULL)
        {
                if (pa->data <= pb->data)
                {
                        pc->next = pa;          //pa小就把pa插在pc后面
                        pc = pc->next;
                        pa = pa->next;
                }
                else
                {
                        pc->next = pb;
                        pc = pc->next;
                        pb = pb->next;
                }

        }
        if (pa != NULL)  //谁有剩余就插在pc后面
        {
                pc->next = pa;
        }
        else if (pb != NULL)
        {
                pc->next = pb;
        }
}


void merge(node* A, node* B, node*& C)    //常规做法：每次new新节点然后连接到链表的最后！
{
	node* pa = A;
	node* pb = B;
	node* pc = C;
	if (pa->data <= pb->data)
	{
		pc->data = pa->data;
		pa = pa->next;
	}
	else
	{
		pc->data = pb->data;
		pb = pb->next;

	}
	while (pa != NULL && pb != NULL)
	{
		if (pa->data < pb->data)
		{
			node* pcc = new node;
			pcc->data = pa->data;
			pcc->next = NULL;
			pc->next = pcc;
			pc = pcc;
			pa = pa->next;
		}
		else
		{
			node* pcc = new node;
			pcc->data = pb->data;
			pcc->next = NULL;
			pc->next = pcc;
			pc = pcc;
			pb = pb->next;
		}
		if (pa != NULL)   //谁有剩余就插在pc后面
		{
			pc->next = pa;
		}
		else if (pb != NULL)
		{
			pc->next = pb;
		}
	}
}
```