双向链表基本操作的 C++代码

```
// 判断链表是否为空，设双向链表类为：DLList
template<class T>
bool DLList::IsEmpty()
{
    return head->right==NULL;
};
```

算法 DLInsert

```
// 在当前结点后插入一个 data 域值为 item 的结点
template<class T>
void DLList::Insert(const T& item)
{
    if(IsEmpty())
    {
        tail=head->right=new DLNode<T>(item, head, NULL);    // head 和 tail 为表头和表尾
指针
        size++;              //size 为表示链表长度
        return;
    }
    //构造结点 p，并使 P 的右指针指向当前结点的右结点，P 的左指针指向当前结点
    DLNode *p=new DLNode<T>(item, currptr, currptr->right);
    currptr->right->left=p;          //currptr 为当前指针，令当前结点的右结点的左指针指
向 P
    currptr->right=p;                //令当前结点的右指针指向 P
    size++;
    if(currptr==tail)    tail=p;        // 若 currptr 是表尾, 令表尾指针指向新插入结点
};

// 在表尾插入一个 data 域值为 item 的结点
template<class T>
void DLList::InsertFromTail(const T &item)
{
    tail=tail->right=new DLNode<T>(item, tail, NULL);
    size++;
};

// 在哨位结点后插入
template<class T>
void DLList::InsertFromHead(const T &item)
{
    if(IsEmpty())
    {
        tail=head->right=new DLNode<T>(item, head, NULL);
        size++;
        return;
    }
    DLNode *p=new DLNode<T>(item, head, head->right);
    head->right->left=p;
    head->right=p;
    size++;
};
```

算法 DeleteNode

```
//删除当前结点并将其 data 值返回给变量 de_item
template<class T>
bool DLList::Delete(T &de_item)
{
    if(IsEmpty()||currptr==head)  return false;    //表为空或当前结点为哨兵结点，则无法删除
    de_item=currptr->data;
    currptr->left->right=currptr->right;
    if(currptr==tail)
        tail=currptr->left;
    else
        currptr->right->left=currptr->left;
    size--;
    currptr=currptr->left;
    delete currptr->right;
    return true;
}
//  删除哨位结点后的第一个真正表结点并将其 data 值返回给变量 de_item
template<class T>
bool DLList::DeleteFromHead(T &de_item)
{
    if(IsEmpty())
    {   cout<<"Empty list!";
        return false;
    }
    DLNode<T> *temp=head->right;
    head->right=temp->right;
    size--;
    de_item=temp->data;
    if(temp==tail) tail=head;              //  若原表中除了哨位结点外只有一个表结点
    delete temp;
    return true;
};

//  删除表尾结点并将其 data 值返回给变量 de_item
template<class T>
bool DLList::DeleteFromTail(T &de_item)
{
    if(IsEmpty())
    {   cout<<"Empty list!";
        return false;
    }
    currptr=tail;
    currptr=currptr->left;                      //  令当前指针指向表尾结点的前驱结点
    de_item=tail->data;
    currptr->right=NULL;
    size--;
    delete tail;
    tail=currptr;
    return true;
};

//删除当前结点的右结点并将其 data 值返回给变量 de_item
template<class T>
bool DLList::DeleteRight(T &de_item)
{
    if(IsEmpty()||currptr->right==NULL)    return false;
```

```
        DLNode<T>*temp=currptr->right;
        de_item=temp->data;
        if(temp==tail) tail=currtpr;
        else  temp->right->left=currptr;
        currptr->right=temp->right;
        delete temp;
        size--;
        return true;
};
```