**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и

автоматизированныхсистем

Лабораторная работа №5

дисциплина: Алгоритмы и структуры данных

тема: «Структуры данных «линейные списки»

Выполнил ст. группы ВТ-22

Проверил: проф. Синюк В.Г.

Белгород 20

**Цель работы:** изучить СД типа «линейный список», научиться их программно реализовывать и использовать.

З а д а н и е

1. Для СД типа «линейный список» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

Динамическая структура

1.1.2. Набор допустимых операций.

-Инициализация

-Деструктор

-Включение элемента за рабочий указатель

-Исключение элемента за рабочим указателем

-Передвинуть рабочий указатель на шаг вперед

-Передвинуть рабочий указатель в начало/конец

-Предикаты список пуст/ указатель в конце

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

Связная (СЛС) или последовательная(ПЛС)

1.2.2. Объем памяти, занимаемый экземпляром СД.

N \* sizeof(elemetOLL)

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

Представлен элементами в динамической памяти ссылающимися друг на друга(СЛС) или массивом(ПЛС) и дескриптором

1.2.4. Характеристику допустимых значений.

CAR(ЛС) = CAR(BaseType)0 + CAR(BaseType)1 +… + CAR(BaseType)max.

1.2.5. Тип доступа к элементам.

-Последовательный(СЛС)

-Прямой(ПЛС)

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

**typedef int** BaseTypeOLL;  
  
*//Элемент списка***typedef struct** elementOLL  
{  
  
 BaseTypeOLL data;  
 **struct** elementOLL \*next;  
  
}elementOLL;  
  
**typedef** elementOLL \*ELptr;  
  
*//Дескриптор списка***typedef struct**{  
 *//Указатель на фиктивный элемент(первый)* ELptr Start;  
 ELptr ptr;  
 **unsigned int** N;  
  
} OLList;

Текст OLL.h

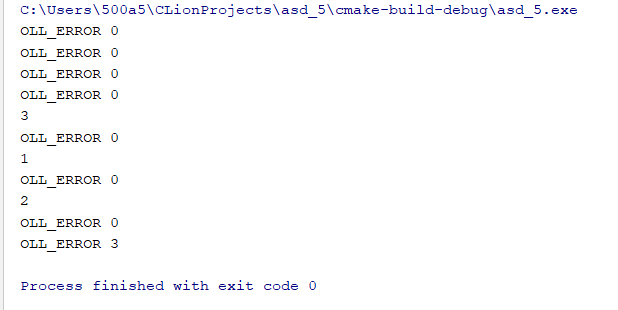
#ifndef **ASD\_5\_OLL\_H**#define **ASD\_5\_OLL\_H**#ifndef **ASD\_OLL\_H**#define **ASD\_OLL\_H**#include **<malloc.h>***//Исключительные ситуации***static const int** OLL\_OK = 0;  
**static const int** OLL\_EMPTY = 1;  
**static const int** OLL\_NOT\_MEM = 2;  
**static const int** OLL\_END = 3;  
  
*//Переменная ошибки***extern short** OLL\_ERROR;  
  
**typedef int** BaseTypeOLL;  
  
*//Элемент списка***typedef struct** elementOLL  
{  
  
 BaseTypeOLL data;  
 **struct** elementOLL \*next;  
  
}elementOLL;  
  
**typedef** elementOLL \*ELptr;  
  
*//Дескриптор списка***typedef struct**{  
 *//Указатель на фиктивный элемент(первый)* ELptr Start;  
 ELptr ptr;  
 **unsigned int** N;  
  
} OLList;  
  
**void** InitOLList(OLList \*L);  
**void** PutOLList(OLList \*L, BaseTypeOLL E);  
**void** GetOLList(OLList \*L, BaseTypeOLL \*E);  
**void** ReadOLList(OLList \*L,BaseTypeOLL \*E);  
**int** EmptyOLList(OLList \*L);  
**int** EndOLList(OLList \*L);  
**unsigned int** Count(OLList \*L);  
**void** BeginPtr(OLList \*L);  
**void** EndPtr(OLList \*L);  
**void** MovePtr(OLList \*L);  
**void** MoveTo(OLList \*L, **unsigned int** n);  
**void** DoneOLList(OLList \*L);  
**void** CopyOLList(OLList \*L1,OLList \*L2);  
  
#endif  
  
#endif *//ASD\_5\_OLL\_H*

Текст OLL.c

#include **<stdlib.h>**#include **"OLL.h"  
  
  
short** OLL\_ERROR;  
  
**void** InitOLList(OLList \*L)  
{  
  
 *//Память под фиктивный элемент* L->Start = malloc(**sizeof**(elementOLL));  
 **if** (L->Start == **NULL**){  
  
 *//Память не выделилась* OLL\_ERROR = OLL\_NOT\_MEM;  
 **return**;  
  
 }  
  
 *//Текущий указатель на начало* L->ptr = L->Start;  
 L->ptr->next = **NULL**;  
  
 *//Инициализация длины списка* L->N = 0;  
  
 OLL\_ERROR = OLL\_OK;  
  
}  
  
**void** PutOLList(OLList \*L, BaseTypeOLL E)  
{  
  
 ELptr new;  
 new = malloc(**sizeof**(elementOLL));  
  
 *//Проверка выделилась ли память под новый узел* **if** (new == **NULL**){  
 OLL\_ERROR = OLL\_NOT\_MEM;  
 **return**;  
 }  
  
 new->data = E;  
  
 *//Указатель на следующий элемент за местом вставки* new->next = L->ptr->next;  
  
 *//Связка предыдущего узла с новым элементом* (L->ptr)->next = new;  
 ++(L->N);  
  
 OLL\_ERROR = OLL\_OK;  
  
}  
  
**void** GetOLList(OLList \*L, BaseTypeOLL \*E)  
{  
  
 *//Если пытаемся удалить из пустого массива* **if** (EmptyOLList(L)){  
 OLL\_ERROR = OLL\_EMPTY;  
 **return**;  
 }  
  
 *//Проверяем не стоит ли указатель в конце* **if**(EndOLList(L)) {  
 OLL\_ERROR = OLL\_END;  
 **return**;  
 }  
  
 elementOLL \*elDel = L->ptr->next;  
  
 *//При удалении необходимо соединить предыдущий элемент со следующим* L->ptr->next = elDel->next;  
 \*E = elDel->data;  
  
 *//Очистим память, уменьшим размер списка* free(elDel);  
 --(L->N);  
  
 OLL\_ERROR = OLL\_OK;  
  
}  
**void** ReadOLList(OLList \*L,BaseTypeOLL \*E)  
{  
  
 *//Если пытаемся читать из пустого массива* **if** (EmptyOLList(L)){  
 OLL\_ERROR = OLL\_EMPTY;  
 **return**;  
 }  
  
 *//Если указатель стоит в конце* **if**(EndOLList(L)) {  
 OLL\_ERROR = OLL\_END;  
 **return**;  
 }  
  
 \*E = L->ptr->next->data;  
  
 OLL\_ERROR = OLL\_OK;  
}  
  
**int** EmptyOLList(OLList \*L)  
{  
  
 **return** L->N == 0;  
  
}  
  
**int** EndOLList(OLList \*L)  
{  
  
 **return** L->ptr->next == **NULL**;  
  
}  
  
**unsigned int** Count(OLList \*L)  
{  
  
 **return** L->N;  
  
}  
  
  
**void** BeginPtr(OLList \*L)  
{  
  
 L->ptr = L->Start;  
  
}  
  
**void** EndPtr(OLList \*L)  
{  
  
 **while** (!EndOLList(L)){  
  
 L->ptr = L->ptr->next;  
  
 }  
  
}  
  
**void** MovePtr(OLList \*L)  
{  
  
 **if**(EndOLList(L)){  
 OLL\_ERROR = OLL\_END;  
 **return**;  
 }  
  
 L->ptr = L->ptr->next;  
  
}  
**void** MoveTo(OLList \*L, **unsigned int** n)  
{  
  
 BeginPtr(L);  
 **int** i = 0;  
  
 **while** ((i<n) && !EndOLList(L)){  
 MovePtr(L);  
 ++i;  
 }  
  
}  
  
  
**void** DoneOLList(OLList \*L)  
{  
  
 BeginPtr(L);  
  
 BaseTypeOLL E;  
  
 *//Пока список не пустой, удаляем элементы* **while** (!EmptyOLList(L)){  
 GetOLList(L, &E);  
 }  
  
 *//освобождаем фиктивный элемент* free(L->ptr);  
  
}  
**void** CopyOLList(OLList \*L1,OLList \*L2)  
{  
  
 *//Удалим лист L2* DoneOLList(L2);  
 *//Инициалиализируем список заново* InitOLList(L2);  
 *//Передвинем указатель L1 в начало* BeginPtr(L1);  
  
 BaseTypeOLL E;  
  
 *//Пока не конец списка, будем добавлять элементы в L1* **while** (!EndOLList(L1)){  
  
 ReadOLList(L1, &E);  
 PutOLList(L2, E);  
 MovePtr(L1);  
 MovePtr(L2);  
  
 }  
  
}

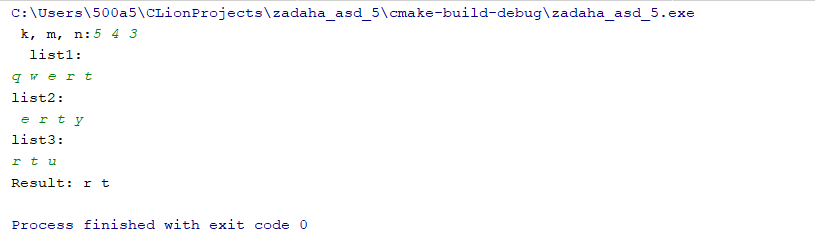
Программа для отладки

#include **<stdio.h>**#include **"OLL.h"  
  
int** main() {  
 OLList list;  
 InitOLList(&list);  
  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
 PutOLList(&list, 1);  
  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
 BeginPtr(&list);  
 PutOLList(&list, 2);  
  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
 EndPtr(&list);  
 PutOLList(&list, 3);  
  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
 BaseTypeOLL E;  
 ReadOLList(&list, &E);  
 printf(**"%d\n"**, E);  
  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
 MoveTo(&list, 1);  
 ReadOLList(&list, &E);  
 printf(**"%d\n"**, E);  
  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
 BeginPtr(&list);  
 GetOLList(&list, &E);  
 printf(**"%d\n"**, E);  
  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
 EndPtr(&list);  
 GetOLList(&list, &E);  
 printf(**"OLL\_ERROR %d\n"**, OLL\_ERROR);  
  
  
 DoneOLList(&list);  
  
  
 **return** 0;  
}



4. Даны натуральные числа k,m,n, последовательности символов s1,...,sk,t1,...,tm,u1,...,un. Получить по одному разу те символы, которые входят во все три последовательности

#include **<stdio.h>**#include **"OLL.h"***// функция заполнения списка из count элементов*OLList\* inputList( OLList \*pt, **int** count )  
{  
 *// cчетчик итерация цикла* **int** i;  
 **char** symbol;  
 *// идем в цикле count раз* **for**( i = 0; i < count; ++i )  
 {  
 scanf(**"%c"**,&symbol);  
 *// и добавляем его в список* PutOLList( pt, symbol );  
 }  
 *// возвращаем указатель на начало списка ( нового списка )* **return** pt;  
}  
  
*// печать списка***void** printList( OLList \*head )  
{  
 *// получаем указатель на начало списка* OLList \*pt = head;  
  
  
  
 *// двигаемся по списку до конца* **while** ( pt != 0 )  
 {  
 *// печатаем текущий элемент* printf(**"%c"**, pt -> ptr) ;  
  
 pt = pt-1;  
 }  
 printf(**"\n"**);  
}  
  
*// функция проверки на вхождение заданного символа в список***int** isExist( OLList \*head, **int** number )  
{  
 *// получаем указатель на голову списка* OLList \*pt = head;  
 *// идем по списку* **while** ( pt != 0 )  
 {  
 *// если текущий элемент равен тому, что нужно найти,  
 // то возвращаем из функции 1* **if** ( pt -> ptr == number )  
 **return** 1;  
 *// двигаемся дальше* pt = pt-1;  
 }  
 *// если прошли весь список и дошли сюда, то нужного элемента в списке нет* **return** 0;  
}  
  
**void** getAloneItems( OLList \*a, OLList \*b, OLList \*c )  
{  
 *// список - результат будет содержать все элементы,  
 // которые встречаются во всех списках* OLList \*result = 0;  
  
 *// идем по первому списку и проверяем на вхождение текущего элемента  
 // во все другие и не вхождение в список - результат, т.к. по одному разу должно быьт выведено* **for**( ; a != 0; a = a -> ptr )  
 {  
 *// если текущий элемент встречается во втором и третьем спсике, но не встречается в результируемом, то  
 // добавляем его в список - результат* **if** ( isExist( b, a -> ptr ) && isExist( c, a -> ptr ) && !isExist( result, a -> ptr ) )  
 PutOLList( result, a -> ptr );  
 }  
  
 *// печатаем результат* printf( **"Result: "**);  
 printList( result );  
}  
  
**int** main() {  
 OLList list;  
 InitOLList(&list);  
 **int** k, m, n;  
 scanf(**"k,m,n: %d%d%d"**,&k,&m,&n);  
 *// зануляем указатели на начало списков* OLList \*sList = 0, \*tList = 0, \*uList = 0;  
*// вводим все данные по спискам*printf(**"list1:\n"**);  
 sList = inputList( sList, k );  
  
 printf(**"list2:\n"**);  
 tList = inputList( tList, m );  
  
 printf(**"list3:\n"**);  
 uList = inputList( uList, n );  
  
 getAloneItems( sList, tList, uList );  
  
  
 **return** 0;  
}



Текст DLL.h

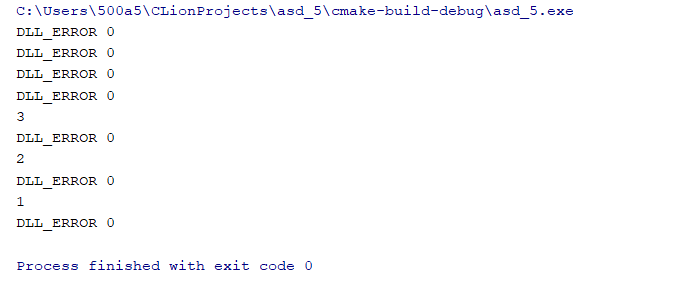
#ifndef **ASD\_5\_DLL\_H**#define **ASD\_5\_DLL\_H**#include **"malloc.h"  
  
static const short** DLL\_OK = 0;  
**static const short** DLL\_EMPTY = 1;  
**static const short** DLL\_NOT\_MEM = 2;  
**static const short** DLL\_END = 3;  
**static const short** DLL\_BEGIN = 4;  
  
**extern short** DLL\_ERROR;  
  
**typedef int** t\_baseDLL;  
  
**typedef struct** elementDLL \*ELptr;  
  
**typedef struct** elementDLL  
{  
  
 t\_baseDLL Data;  
 ELptr Llink;  
 ELptr Rlink;  
  
}elementDLL;  
  
**typedef struct**{  
  
 ELptr first;  
 ELptr last;  
 ELptr ptr;  
 **int** size;  
  
}DLL;  
  
**void** initDLList(DLL \*list);  
**void** putDLListPred(DLL \*list, t\_baseDLL E);  
**void** putDLListPost(DLL \*list, t\_baseDLL E);  
  
**void** getDLListPred(DLL \*list, t\_baseDLL \*E);  
**void** getDLListPost(DLL \*list, t\_baseDLL \*E);  
  
  
**void** DLListMoveL(DLL \*list);  
**void** DLListMoveR(DLL \*list);  
  
**short** DLListEmpty(DLL \*list);  
**short** DLListBegin(DLL \*list);  
**short** DLListEnd(DLL \*list);  
  
**void** DLListPtrEnd(DLL \*list);  
**void** DLListPtrBegin(DLL \*list);  
  
**void** DLListDone(DLL \*list);  
  
#endif *//ASD\_5\_DLL\_H*

Текст DLL.c

#include **<stdlib.h>**#include **"DLL.h"  
  
short** DLL\_ERROR;  
  
**void** initDLList(DLL \*list)  
{  
 list->last = **NULL**;  
 list->first = **NULL**;  
 list->ptr = **NULL**;  
 list->size = 0;  
 DLL\_ERROR = DLL\_OK;  
}  
**void** putDLListPred(DLL \*list, t\_baseDLL E)  
{  
 ELptr newEl = malloc(**sizeof**(elementDLL));  
 **if**(newEl == **NULL**){  
 DLL\_ERROR = DLL\_NOT\_MEM;  
 **return**;  
 }  
 newEl->Data = E;  
 **if**(DLListEmpty(list)){  
 list->first = newEl;  
 list->last = newEl;  
 list->ptr = newEl;  
 newEl->Llink = **NULL**;  
 newEl->Rlink = **NULL**;  
 }**else**{  
 newEl->Llink = list->ptr->Llink;  
 newEl->Rlink = list->ptr;  
 list->ptr->Llink = newEl;  
 **if**(newEl->Llink == **NULL**)  
 list->first = newEl;  
 }  
 DLL\_ERROR = DLL\_OK;  
 ++(list->size);  
}  
**void** putDLListPost(DLL \*list, t\_baseDLL E)  
{  
 ELptr newEl = malloc(**sizeof**(elementDLL));  
 **if**(newEl == **NULL**){  
 DLL\_ERROR = DLL\_NOT\_MEM;  
 **return**;  
 }  
 newEl->Data = E;  
 **if**(DLListEmpty(list)){  
 list->first = newEl;  
 list->last = newEl;  
 list->ptr = newEl;  
 newEl->Llink = **NULL**;  
 newEl->Rlink = **NULL**;  
 }**else**{  
 newEl->Rlink = list->ptr->Rlink;  
 newEl->Llink = list->ptr;  
 list->ptr->Rlink = newEl;  
 **if**(newEl->Rlink == **NULL**)  
 list->last = newEl;  
 }  
 DLL\_ERROR = DLL\_OK;  
 ++(list->size);  
}  
  
**void** getDLListPred(DLL \*list, t\_baseDLL \*E)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return**;  
 }  
 **if**(list->size == 1){  
 \*E = list->ptr->Data;  
 free(list->ptr);  
 list->last = **NULL**;  
 list->first = **NULL**;  
 --(list->size);  
 **return**;  
 }  
 **if**(DLListBegin(list)){  
 DLL\_ERROR = DLL\_BEGIN;  
 **return**;  
 }  
 ELptr delEl = list->ptr->Llink;  
 list->ptr->Llink = delEl->Llink;  
 **if**(list->ptr->Llink == **NULL**)  
 list->first = list->ptr;  
 **else** delEl->Llink->Rlink = list->ptr;  
 \*E = delEl->Data;  
 free(delEl);  
 DLL\_ERROR = DLL\_OK;  
 --(list->size);  
}  
**void** getDLListPost(DLL \*list, t\_baseDLL \*E)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return**;  
 }  
 **if**(list->size == 1){  
 \*E = list->ptr->Data;  
 free(list->ptr);  
 list->last = **NULL**;  
 list->first = **NULL**;  
 --(list->size);  
 **return**;  
 }  
 **if**(DLListEnd(list)){  
 DLL\_ERROR = DLL\_END;  
 **return**;  
 }  
 ELptr delEl = list->ptr->Rlink;  
 list->ptr->Rlink = delEl->Rlink;  
 **if**(list->ptr->Rlink == **NULL**)  
 list->last = list->ptr;  
 **else** delEl->Rlink->Llink = list->ptr;  
 \*E = delEl->Data;  
 free(delEl);  
 DLL\_ERROR = DLL\_OK;  
 --(list->size);  
}  
  
**void** DLListMoveL(DLL \*list)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return**;  
 }  
 **if**(DLListBegin(list)){  
 DLL\_ERROR = DLL\_BEGIN;  
 **return**;  
 }  
 list->ptr = list->ptr->Llink;  
 DLL\_ERROR = DLL\_OK;  
}  
**void** DLListMoveR(DLL \*list)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return**;  
 }  
 **if**(DLListEnd(list)){  
 DLL\_ERROR = DLL\_END;  
 **return**;  
 }  
 list->ptr = list->ptr->Rlink;  
 DLL\_ERROR = DLL\_OK;  
}  
**short** DLListEmpty(DLL \*list)  
{  
 **return** list->size == 0;  
}  
**short** DLListBegin(DLL \*list)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return** 1;  
 }  
 **return** list->ptr->Llink == **NULL**;  
}  
**short** DLListEnd(DLL \*list)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return** 1;  
 }  
 **return** list->ptr->Rlink == **NULL**;  
}  
**void** DLListPtrEnd(DLL \*list)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return**;  
 }  
 list->ptr = list->last;  
 DLL\_ERROR = DLL\_OK;  
}  
  
  
**void** DLListPtrBegin(DLL \*list)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return**;  
 }  
 list->ptr = list->first;  
 DLL\_ERROR = DLL\_OK;  
}  
**void** DLListDone(DLL \*list)  
{  
 **if**(DLListEmpty(list)){  
 DLL\_ERROR = DLL\_EMPTY;  
 **return**;  
 }  
 DLListPtrBegin(list);  
 t\_baseDLL E;  
 **while** (DLListEnd(list)){  
 getDLListPost(list, &E);  
 }  
 free(list->ptr);  
 list->ptr = **NULL**;  
 list->last = **NULL**;  
 list->first = **NULL**;  
 DLL\_ERROR = DLL\_OK;  
}

Текст программы для отладки

#include **<stdio.h>**#include **"DLL.h"  
  
int** main() {  
  
 DLL list;  
 t\_baseDLL E;  
 initDLList(&list);  
  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 putDLListPost(&list, 3);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 putDLListPost(&list, 2);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 putDLListPred(&list, 1);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 DLListPtrEnd(&list);  
 getDLListPred(&list, &E);  
 printf(**"%d\n"**, E);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 DLListPtrBegin(&list);  
 getDLListPost(&list, &E);  
 printf(**"%d\n"**, E);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 getDLListPost(&list, &E);  
 printf(**"%d\n"**, E);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
  
 DLListDone(&list);  
  
 **return** 0;  
}



Текст Deck.h

#ifndef **ASD\_DECK\_H**#define **ASD\_DECK\_H**#include **"DLL.h"  
  
static const short** DECK\_OK = **DLL\_OK**;  
**static const short** DECK\_NOT\_MEM = **DLL\_NOT\_MEM**;  
**static const short** DECK\_EMPTY = **DLL\_EMPTY**;  
  
**extern short** DECK\_ERROR;  
  
**typedef** DLL Deck;  
**typedef** t\_baseDLL t\_baseDeck;  
  
**void** initDeck(Deck \*deck);  
**void** doneDeck(Deck \*deck);  
  
**void** PutEnd(Deck \*deck, t\_baseDLL E);  
**void** PutBegin(Deck \*deck, t\_baseDLL E);  
  
**void** GetEnd(Deck \*deck, t\_baseDLL \*E);  
**void** GetBegin(Deck \*deck, t\_baseDLL \*E);  
  
**short** EmptyDeck(Deck deck);  
  
#endif

Текст Deck.c

#include **"../Headers/Deck.h"  
  
short** DECK\_ERROR;  
  
**void** initDeck(Deck \*deck)  
{  
 initDLList(deck);  
 DECK\_ERROR = DLL\_ERROR;  
}  
**void** doneDeck(Deck \*deck)  
{  
 DLListDone(deck);  
 DECK\_ERROR = DLL\_ERROR;  
}  
  
**void** PutEnd(Deck \*deck, t\_baseDLL E)  
{  
 DLListPtrEnd(deck);  
 putDLListPost(deck, E);  
 DLL\_ERROR = DLL\_ERROR;  
}  
**void** PutBegin(Deck \*deck, t\_baseDLL E)  
{  
 DLListPtrBegin(deck);  
 putDLListPred(deck, E);  
 DLL\_ERROR = DLL\_ERROR;  
}  
  
**void** GetEnd(Deck \*deck, t\_baseDLL \*E)  
{  
 DLListPtrEnd(deck);  
 DLListMoveL(deck);  
 getDLListPost(deck, E);  
 DECK\_ERROR = DLL\_ERROR;  
}  
**void** GetBegin(Deck \*deck, t\_baseDLL \*E)  
{  
 DLListPtrBegin(deck);  
 DLListMoveR(deck);  
 getDLListPred(deck, E);  
 DECK\_ERROR = DLL\_ERROR;  
}  
  
**short** EmptyDeck(Deck deck)  
{  
 **return** DLListEmpty(&deck);  
}

Текст программы для отладки

**int** main() {  
  
 Deck deck;  
 t\_baseDeck E;  
 initDeck(&deck);  
 printf(**"DECK\_ERROR %d\n"**, DECK\_ERROR);  
  
 PutBegin(&deck, 1);  
 printf(**"DECK\_ERROR %d\n"**, DECK\_ERROR);  
  
 PutBegin(&deck, 2);  
 printf(**"DECK\_ERROR %d\n"**, DECK\_ERROR);  
  
 PutBegin(&deck, 3);  
 printf(**"DECK\_ERROR %d\n"**, DECK\_ERROR);  
  
 PutEnd(&deck, 0);  
 printf(**"DECK\_ERROR %d\n"**, DECK\_ERROR);  
  
 **while** (!EmptyDeck(deck)){  
 GetEnd(&deck, &E);  
 printf(**"%d\n"**, E);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 }  
  
 GetEnd(&deck, &E);  
 printf(**"%d\n"**, E);  
 printf(**"DLL\_ERROR %d\n"**, DLL\_ERROR);  
  
 doneDeck(&deck);  
  
 **return** 0;  
}

Выходные данные

