МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
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 автоматизированных систем

Лабораторная работа №5

по дисциплине: Алгоритмы и структуры данных
тема: Структуры данных «линейные списки» (Pascal/C)»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили: асс. Солонченко Роман
Евгеньевич

Белгород 2023г.

Лабораторная работа №5
Структуры данных «линейные списки» (Pascal/C)»
Вариант 10

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

1. Для СД типа «линейный список» определить:

   1.1. Абстрактный уровень представления СД:

      1.1.1. Характер организованности и изменчивости.
      Характер организованности - **линейный**. Характер изменчивости - **динамический**.

      1.1.2. Набор допустимых операций.
      Инициализация, включение элемента, исключение элемента, чтение текущего элемента, переход в начало списка, переход в конец списка, переход к следующему элементу, переход к i-му элементу, определение длины списка, уничтожение списка.

   1.2. Физический уровень представления СД:

      1.2.1. Схему хранения.
      Схема хранения - **связный.**

      1.2.2. Объем памяти, занимаемый экземпляром СД.
      Размер СД состоит из размера дескриптора, размера фиктивного элемента и размера всех элементов. Размер дескриптора: указатель на начало (машинное слово - 4 байт), рабочий указатель (машинное слово - 4 байт) и количество элементов (int - 4 байт). Размер элементов: $1+N\cdot(sizeof(BaseType)$ 4) байт. $V = 13 + N \cdot (sizeof(BaseType) + 4)$

      1.2.3. Формат внутреннего представления СД и способ его интерпретации.
      Дескриптор находится в статической памяти, элементы ОЛС находятся в динамической памяти.

      1.2.4. Характеристику допустимых значений.
      $Car(C) = 1+Car(BaseType)+Car(BaseType)^2+...+Car(BaseType)^{max}$.

      1.2.5. Тип доступа к элементам.
      Тип доступа к элементам - **последовательный**.

   1.3. Логический уровень представления СД.

      1.3.1. Способ описания СД и экземпляра СД на языке программирования.

```
List l;
InitList(&l);
```

2. Реализовать СД типа «линейный список» в соответствии с вариантом индивидуального задания (см. табл.14) в виде модуля.
   main.c (тесты)

```c
#include <algc.h>

#include <stdio.h>
#include <assert.h>
#include <string.h>

void testPutList() {
    List t1;
    InitList(&t1);
    PutList(&t1, 1);
    int e;
    ReadList(&t1, &e);
    assert(ListError == ListOk &&
    t1.N == 1 &&
    e == 1);

    PutList(&t1, 2);
    ReadList(&t1, &e);
    assert(ListError == ListOk &&
    t1.N == 2 &&
    e == 2);

    MovePtr(&t1);
    MovePtr(&t1);
    PutList(&t1, 3);
    ReadList(&t1, &e);
    assert(ListError == ListOk &&
    t1.N == 3 &&
    e == 3);

    DoneList(&t1);
}

void testGetList() {
    List t1;
    InitList(&t1);
    PutList(&t1, 1);
    MovePtr(&t1);
    PutList(&t1, 2);
    MovePtr(&t1);
    PutList(&t1, 3);
    int e;
```

```
    EndPtr(&t1);
    GetList(&t1, &e);
    assert(ListError == ListEnd);

    BeginPtr(&t1);
    MovePtr(&t1);
    MovePtr(&t1);
    GetList(&t1, &e);
    assert(ListError == ListOk && e == 3);

    BeginPtr(&t1);
    MovePtr(&t1);
    GetList(&t1, &e);
    assert(ListError == ListOk && e == 2);

    BeginPtr(&t1);
    GetList(&t1, &e);
    assert(ListError == ListOk && e == 1);

    GetList(&t1, &e);
    assert(ListError == ListUnder || ListError == ListEnd);

    DoneList(&t1);
}

void testReadList() {
    List t1;
    InitList(&t1);
    PutList(&t1, 1);
    MovePtr(&t1);
    PutList(&t1, 2);
    MovePtr(&t1);
    PutList(&t1, 3);
    int e;

    EndPtr(&t1);
    ReadList(&t1, &e);
    assert(ListError == ListEnd);

    BeginPtr(&t1);
    MovePtr(&t1);
    MovePtr(&t1);
    ReadList(&t1, &e);
```

```
    assert(ListError == ListOk && e == 3);

    BeginPtr(&t1);
    MovePtr(&t1);
    ReadList(&t1, &e);
    assert(ListError == ListOk && e == 2);

    BeginPtr(&t1);
    ReadList(&t1, &e);
    assert(ListError == ListOk && e == 1);

    DoneList(&t1);
}

void testEndList() {
    List t1;
    InitList(&t1);
    EndList(&t1);
    assert(ListError == ListOk);

    PutList(&t1, 1);
    PutList(&t1, 2);
    PutList(&t1, 3);
    BeginPtr(&t1);
    assert(ListError == ListOk && !EndList(&t1));
    MovePtr(&t1);
    assert(ListError == ListOk && !EndList(&t1));
    MovePtr(&t1);
    assert(ListError == ListOk && !EndList(&t1));
    MovePtr(&t1);
    assert(ListError == ListOk && EndList(&t1));

    DoneList(&t1);
}

void testCount() {
    List t1;
    InitList(&t1);
    assert(Count(&t1) == 0 && ListError == ListOk);
    PutList(&t1, 1);
    assert(Count(&t1) == 1 && ListError == ListOk);
    PutList(&t1, 2);
    assert(Count(&t1) == 2 && ListError == ListOk);
```

```
    PutList(&t1, 3);
    assert(Count(&t1) == 3 && ListError == ListOk);

    DoneList(&t1);
}

void testBeginPtr() {
    List t1;
    InitList(&t1);
    PutList(&t1, 1);
    PutList(&t1, 2);
    MovePtr(&t1);
    PutList(&t1, 3);
    BeginPtr(&t1);

    assert(t1.ptr == t1.Start && ListError == ListOk);

    DoneList(&t1);
}

void testEndPtr() {
    List t1;
    InitList(&t1);

    PutList(&t1, 1);
    MovePtr(&t1);
    PutList(&t1, 2);
    MovePtr(&t1);
    PutList(&t1, 3);
    EndPtr(&t1);

    assert(t1.ptr->data == 3 && ListError == ListOk);

    DoneList(&t1);
}

void testMovePtr() {
    List t1;
    InitList(&t1);
    MovePtr(&t1);
    assert(ListError == ListEnd);

    PutList(&t1, 1);
```

```
    MovePtr(&t1);
    assert(ListError == ListOk && t1.ptr->data == 1);
    PutList(&t1, 2);
    MovePtr(&t1);
    assert(ListError == ListOk && t1.ptr->data == 2);

    PutList(&t1, 3);
    MovePtr(&t1);
    assert(ListError == ListOk && t1.ptr->data == 3);

    DoneList(&t1);
}

void testMoveTo() {
    List t1;
    InitList(&t1);
    MoveTo(&t1, 177);
    assert(ListError == ListEnd);

    PutList(&t1, 1);
    MovePtr(&t1);
    PutList(&t1, 2);
    MovePtr(&t1);
    PutList(&t1, 3);

    MoveTo(&t1, 0);
    assert(ListError == ListOk && t1.ptr->data == 1);
    MoveTo(&t1, 2);
    assert(ListError == ListOk && t1.ptr->data == 3);
    MoveTo(&t1, 1);
    assert(ListError == ListOk && t1.ptr->data == 2);

    DoneList(&t1);
}

void test() {
    testPutList();
    testGetList();
    testReadList();
    testEndList();
    testCount();
    testBeginPtr();
    testEndPtr();
```

```
    testMovePtr();
    testMoveTo();
}

int main() {
    test();

    return 0;
}
```

alg.h (заголовки)

```
#ifndef SINGLY_CONNECTED_LIST
#define SINGLY_CONNECTED_LIST

#define ListOk 0
#define ListNotMem 1
#define ListUnder 2
#define ListEnd 3

#ifndef CUSTOM_BASE_TYPE
typedef int BaseType;
#endif

typedef struct element_ {
    BaseType data;
    struct element_ * next;
} element;
typedef element* ptrel;

typedef struct {
    ptrel Start;
    ptrel ptr;
    unsigned int N;
} List;

extern int ListError;

void InitList(List *L);
void PutList(List *L, BaseType E);
void GetList(List *L, BaseType *E);
void ReadList(List *L,BaseType *E);
int FullList(List *L);
```

```
int EndList(List *L);
unsigned int Count(List *L);
void BeginPtr(List *L);
void EndPtr(List *L);
void MovePtr(List *L);
void MoveTo(List *L, unsigned int n);
void DoneList(List *L);
void CopyList(List *L1,List *L2);

#endif
```

task2.c (реализации функций)

```
#include <lab5/singlyconnectedlist.h>

#include <stddef.h>
#include <malloc.h>

int ListError = ListOk;

void InitList(List *L) {
    ptrel newElement = malloc(sizeof(element));
    if (newElement == NULL) {
        ListError = ListNotMem;
        return;
    }
    newElement->next = NULL;

    L->Start = newElement;
    L->ptr = newElement;
    L->N = 0;
}

void PutList(List *L, BaseType E) {
    ptrel newElement = malloc(sizeof(element));
    if (newElement == NULL) {
        ListError = ListNotMem;
        return;
    }

    newElement->data = E;
    newElement->next = NULL;
```

```
    ptrel currentElement = L->ptr;
    ptrel nextElement = currentElement->next;
    currentElement->next = newElement;
    newElement->next = nextElement;
    L->N++;
    ListError = ListOk;
}

void GetList(List *L, BaseType *E) {
    if (Count(L) == 0) {
        ListError = ListUnder;
        return;
    }

    if (EndList(L)) {
        ListError = ListEnd;
        return;
    }

    ptrel currentElement = L->ptr;
    *E = currentElement->next->data;
    L->N--;
    ptrel nextNextElement = currentElement->next->next;
    free(currentElement->next);
    currentElement->next = nextNextElement;
    ListError = ListOk;
}

void ReadList(List *L,BaseType *E) {
    if (Count(L) == 0) {
        ListError = ListUnder;
        return;
    }

    if (EndList(L)) {
        ListError = ListEnd;
        return;
    }

    ptrel currentElement = L->ptr;
    *E = currentElement->next->data;
    ListError = ListOk;
}
```

```c
// Зачем эта функция???
int FullList(List *L) {
    ListError = ListOk;
    return 0;
}

int EndList(List *L) {
    ListError = ListOk;
    return L->ptr->next == NULL;
}

unsigned int Count(List *L) {
    ListError = ListOk;
    return L->N;
}

void BeginPtr(List *L) {
    ListError = ListOk;
    L->ptr = L->Start;
}

void EndPtr(List *L) {
    L->ptr = L->Start;

    while (L->ptr->next != NULL) {
        L->ptr = L->ptr->next;
    }
    ListError = ListOk;
}

void MovePtr(List *L) {
    if (EndList(L)) {
        ListError = ListEnd;
        return;
    }

    L->ptr = L->ptr->next;
    ListError = ListOk;
}

void MoveTo(List *L, unsigned int n) {
    BeginPtr(L);
```

```
    for (int i = 0; i < n + 1; i++) {
        if (L->ptr->next == NULL) {
            ListError = ListEnd;
            return;
        }

        MovePtr(L);
    }
    ListError = ListOk;
}

static void freeElement(ptrel element) {
    if (element == NULL)
        return;

    freeElement(element->next);
    free(element);
}

void DoneList(List *L) {
    EndPtr(L);
    if (ListError != ListOk)
        return;

    freeElement(L->Start);
    L->Start = NULL;
    L->ptr = NULL;
    L->N = 0;
    ListError = ListOk;
}

void CopyList(List *L1,List *L2) {
    do
    {
        PutList(L2, L1->ptr->data);
        if (EndList(L1)) break;

        MovePtr(L1);
    } while (ListError == ListOk);
}
```

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания с использованием модуля, полученного в результате выполнения пункта 2 задания.
main.c (основная программа)

```c
#include <algc.h>

#include <stdio.h>

int main() {
    List l;
    InitList(&l);
    if (ListError != ListOk) {
        printf("An error occured trying to init a list, code: %d", ListError);
        return ListError;
    }

    int amount;
    printf("Enter elements amount: ");
    scanf("%d", &amount);
    printf("Enter elements: ");
    for (int i = 0; i < amount; i++) {
        int element;
        scanf("%d", &element);

        PutList(&l, element);
        if (ListError != ListOk) {
            printf("An error occured trying to insert an element in list, code: %d",
            ↪  ListError);
            return ListError;
        }

        MovePtr(&l);
        if (ListError != ListOk) {
            printf("An error occured trying to insert an element in list, code: %d",
            ↪  ListError);
            return ListError;
        }
    }

    int step;
    printf("Enter step: ");
    scanf("%d", &step);
```

```
    int previousElement;
    BeginPtr(&l);
    GetList(&l, &previousElement);
    if (ListError != ListOk) {
        printf("An error occured trying to get an element in list, code: %d", ListError);
        return ListError;
    }

    for (int i = 1; i < amount; i++) {
        int currentElement;
        GetList(&l, &currentElement);
        if (ListError != ListOk) {
            printf("An error occured trying to get an element in list, code: %d",
            ↪  ListError);
            return ListError;
        }

        if (currentElement - previousElement != step) {
            printf("A condition is breached with elements %d and %d", previousElement,
            ↪  currentElement);
            return 0;
        }

        previousElement = currentElement;
    }


    printf("A condition is fulfilled");

    return 0;
}
```

Вывод: в ходе лабораторной работы изучили СД типа «линейный список», научились их программно реализовывать и использовать.