Для разворота строки через стек необходимо сначала реализовать СД типа стек. Стек, как и линейный список, представляет собой на абстрактном уровне линейную последовательность элементов, а потому СД «стек» может быть реализована на базе линейного списка. В заголовочном файле объявим константы – коды ошибок; их множество будет подмножеством кодов ошибок при работе с линейным списком. Определим тип Stack (стек) как переименование типа List (ОЛС на массиве). Далее объявим прототипы функций для работы со стеком. Их будет значительно меньше, чем функций для работы со списком, поскольку для стека отпадает необходимость функций перемещения по нему рабочего указателя – включение, исключение и чтение проводятся только с вершины.

```
#ifndef __STACK_5_H
#define __STACK_5_H
#include "../АСД 5 си/list.h"

const short StackOk = ListOk;
const short StackUnder = ListUnder;
const short StackOver = ListNotMem;
int StackError; // Переменная ошибок
typedef List Stack;

// Инициализация стека по адресу s
void InitStack(Stack *s);
//Поместить элемент со значением E в стек по адресу s
void PutStack(Stack *s, BaseType E);
//Извлечь элемент из стека по адресу s в переменную по адресу E
void GetStack(Stack *s, BaseType *E);
// Проверка, пуст ли стек по адресу s
int  EmptyStack(Stack *s);
//Прочитать элемент из вершины стека по адресу s и сохранить в
переменную по адресу E
void ReadStack(Stack *s, BaseType *E);
//Уничтожить стек по адресу s
void DoneStack(Stack *s);

#endif
```

Задача большей части функций для работы со списком сводится к вызову аналогичной функции для линейного списка, на который отображен данный стек. Исключение – функции включения и исключения элемента, которые должны также контролировать положение рабочего указателя таким образом, чтобы он всегда указывал на «вершину» стека. Будем считать вершиной первый элемент. Тогда пусть рабочий указатель по умолчанию указывает на начало списка свободных элементов в массиве, хранящем элементы линейных списков. При таком положении включение элемента всегда будет происходить в начало

стека (включенный элемент станет новым первым в этом стеке). Включение будет происходить без проблем: все, что требуется – после вызова функции PutList опять установить указатель на нулевой элемент массива (начало ССЭ). Но чтобы исключить первый элемент из стека, рабочий указатель списка должен указывать на первый элемент. Тогда перед вызовом функции GetList необходимо установить указатель на этот первый элемент (рабочий указатель принимает значение указателя на начало списка), а после вызова – установить его на начало ССЭ.

```
#ifndef __STACK_5_C
#define __STACK_5_C
#include "Stack_5.h"
#include "../АСД 5 си/list.c"

void InitStack(Stack *s) {
    InitList(s);
}
void PutStack(Stack *s, BaseType E) {
    PutList(s, E);
    s->ptr = 0;
}
void GetStack(Stack *s, BaseType *E) {
    s->ptr = s->start;
    GetList(s, E);
    s->ptr = 0;
}
int EmptyStack(Stack *s) {
    return !FullList(s);
}
void ReadStack(Stack *s, BaseType *E) {
    ReadList(s, E);
}
void DoneStack(Stack *s) {
    DoneList(s);
}

#endif
```

Завершив реализацию функций для работы со стеком, мы можем написать функцию, решающую поставленную задачу – разворот строки через стек и запись результата в массив char, выделенной памяти под который достаточно для хранения строки, равной по размеру исходной.

```
//Записывает по адресу result зеркальное отржение строки по адресу
source
```

```
void *stringReversal (char *source, char *result) {
    char *source_ptr = source;
    Stack for_reversal;
    InitStack(&for_reversal);
    while (*source_ptr != '\0') {
        int cur_el = (int) (*source_ptr);
        PutStack(&for_reversal, cur_el);
        source_ptr++;
    }
    char *result_ptr = result;
    while (!EmptyStack(&for_reversal)) {
        int cur_el;
        GetStack(&for_reversal, &cur_el);
        (*result_ptr) = (char) cur_el;
        result_ptr++;
    }
    *result_ptr = '\0';
}
```

А реализовав функцию для сравнения строк, мы можем написать несколько тестов для этой функции и убедиться, что она выполняет поставленную задачу.

```
//возвращает отрицательное значение, если строка lhs располагается
до rhs в лексикографическом порядке (как в словаре),
//значение 0, если lhs и rhs равны, иначе - положительное
значение.
int strcomp(const char *lhs, const char *rhs) {
    while (*lhs != '\0' && *lhs == *rhs) {
        lhs++;
        rhs++;
    }
    return *lhs - *rhs;
}
```

```
void Test_stringReversal_Empty() {
    char *s = "";
    char r[1];
    char *model_r = "";
    stringReversal(s, r);
    assert(strcomp(r, model_r) == 0);
}
void Test_stringReversal_SomeText() {
    char *s= "ich liebe dich";
    char r[16];
    char *model_r = "hcid ebeil hci";
    stringReversal(s, r);
    assert(strcomp(r, model_r) == 0);
```

```
}
void Test_stringReversal_Equal() {
    char *s = "dennis sinned";
    char r[16];
    char *model_r = "dennis sinned";
    stringReversal(s, r);
    assert(strcomp(r, model_r) == 0);
}

int main() {
    InitMem();
    Test_stringReversal_Empty();
    Test_stringReversal_SomeText();
    Test_stringReversal_Equal();
    printf("All is OK!");
}
```

```
"C:\Users\sovac\Desktop\ASD_third_semester\АСД 5 си — контрольное задание\Stack_Task.exe"
All is OK!
Process finished with exit code 0
```