**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 №2

дисциплина: Алгоритмы и структуры данных

тема: «Производные структуры данных. Структура данных «строка» (Pascal/C)»

Выполнил ст. группы

Проверил: проф. Синюк В.Г.

Белгород 2

Цель работы: изучение встроенной структуры данных тип «Строка», разработка и использование производных структур данных строкового типа.

Вариант 4

**З а д а н и е**

1. Для СД типа строка определить:

1. Абстрактный уровень представления СД:

1.1. Характер организованности и изменчивости.

1.2. Набор допустимых операций.

2. Физический уровень представления СД:

2.1. Схему хранения.

2.2. Объем памяти, занимаемый экземпляром СД.

2.3. Формат внутреннего представления СД и способ его интерпретации.

2.4. Характеристику допустимых значений.

2.5. Тип доступа к элементам.

3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД строкового типа в соответствии с вариантом индивидуального задания в виде модуля. Определить и обработать исключительные ситуации.

3.Разработать программу для решения задачи в соответствии с вариантом индивидуального задания с использованием модуля, полученного в результате выполнения пункта 2.

4. Заголовок: function LastPost(s1,s2:string):word/ unsigned LastPost(string s1,string s2).

Назначение: поиск последнего вхождения подстроки s2 в строку s1

Входные параметры: s1,s2.

Выходные параметры: нет.

Реализация на языке C:

#if !defined(\_\_FORM4\_H)

#define \_\_FORM4\_H

const ...; // Определение исключительных ситуаций

typedef struct str

{

char s[1024];

unsigned N; // Динамическая длина строки

};

typedef str \*string1;

void WriteToStr(string1 st, char \*s);

void WriteFromStr(char \*s, string1 st);

void InputStr(string1 st);

void OutputStr(string1 st);

int Comp(string1 s1, string1 s2);

void Delete(string1 s, unsigned Index, unsigned Count);

void Insert(string1 Subs, string1 s, unsigned Index);

void Concat(string1 s1, string1 s2, string1 srez);

void Copy(string1 s, unsigned Index, unsigned Count, string1 Subs);

unsigned Length(string1 s);

unsigned Pos(string1 SubS, string1 s);

int StrError; // Переменная ошибок

//...

#endif

1. Для СД типа строка определить:

1.1. Абстрактный уровень представления СД:

1.1.1 Характер организованности и изменчивости.

Линейная структура(последовательность), динамическая структура

1.1.2. Набор допустимых операций.

В Си:

Присваивание, Сравнение, Конкатенация.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

Последовательная

1.2.2. Объем памяти, занимаемый экземпляром СД.

V = K + 1 где

V объем строки в байтах

K максимальное количество элементов

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

Массив элементов типа char, которые нумеруются от 1 до k, заканчивается нулевым символом '\0', элемент, с индексом K равен 0 как признак конца строки. ASCI коды символов интерпретируются в двоичный код.

1.2.4. Характеристику допустимых значений.

Количество допустимых значений СД типа строка = CAR(string) = 1 + 2561 + 2562 + …+256K, где K – максимальное число элементов в строке

1.2.5. Тип доступа к элементам.

Прямой

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

Способ описания –

//Если экземпляр типа строка в статической памяти и занимает N байт

#define N …//значение

char s[N];

Экземпляр -

#define N 10

char s[N];

Способ описания –

//Если экземпляр типа строка в статической памяти и занимает N (значение) байт

typedef char t\_str …//значение;

t\_str s;

Экземпляр

typedef char t\_str[10]

t\_str s;

Способ описания –

//Если экземпляр типа строка в динамической памяти после обращении к процедуре выделения памяти alloc или malloc

typedef char\*t\_str;

t\_str //переменная;

Экземпляр

typedef char \*t\_str;

t\_str s;

Способ описания –

//Если экземпляр типа строка в динамической памяти после обращении к процедуре выделения памяти new. Занимает N байт, адрес пишется в переменную (//переменная)

typedef char t\_str[N];

typedef t\_str\* p\_str

//переменная = (p\_str) new p\_str

p\_str = //переменная;

Экземпляр

typedef char t\_str[10];

typedef t\_str\* p\_str

ps = (p\_str) new p\_str

p\_str = ps;

Заголовочный файл

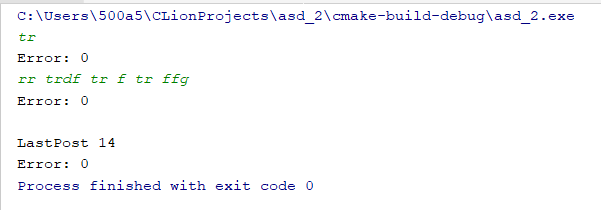
#ifndef **ASD\_2\_STR\_H**#define **ASD\_2\_STR\_H**#include **<malloc.h>**#include **<stdio.h>***//Операция прошла успешно***static const short** STR\_SUCCESSFUL= 0;  
  
*//Выход за границу максимально разрешенного размера строки  
//при вводе в нее данных***static const short** STRING\_INPUT\_ERROR = -1;  
  
  
*//Выход за границу максимально разрешенного размера строки  
//при вставке данных из одной строки в другую***static const short** STRING\_INSERT\_ERROR = -2;  
  
*//Попытка вставить элемент на место которое не существует***static const short** STRING\_NO\_PLACE = -3;  
  
*//Выход за границу максимально разрешенного размера строки  
//при конкатенации в нее данных***static const short** STRING\_CONCATEN\_ERROR = -4;  
*// ошибка поиска в меньшей строки большей подстроки***static const short** STRING\_Pos\_ERROR = -5;  
  
**extern short** STRING\_ERROR;  
#define **M** 1024  
**typedef struct**{  
 **char** s[**M**];  
 **unsigned** N; *// Динамическая (текущая) длина строки*} str;  
  
**typedef** str string1[2];  
  
*//Запись данных в строку st из строки s  
//Строка s заканчивается нулевым символом***void** WriteToStr(string1 st, **char** \*s);  
  
*//Запись данных в строку s из строки st  
//Строка s заканчивается нулевым символом***void** WriteFromStr(**char** \*s, string1 st);  
  
*//Ввод строки s с клавиатуры***void** InputStr(string1 st);  
  
*//Вывод строки s на экран монитора***void** OutputStr(string1 st);  
  
*//Сравнивает строки s1 и s2 возвращает 0 если  
//s1 == s2; 1 если s1 > s2; -1 если s1 < s2***int** Comp(string1 s1, string1 s2);  
  
*//Удаляет count символов из строки s  
//начиная с позиции index***void** Delete(string1 st, **unsigned** index, **unsigned** count);  
  
*//Вставляет подстроку subS в строку s  
//начиная с позиции index***void** Insert(string1 subS, string1 st, **unsigned** index);  
  
*//Выполняет конкатенацию строк s1 и s2 результат помещает в sRez***void** Concat(string1 s1, string1 s2, string1 sRez);  
  
*//Записывает count символов в строку subS из строки s  
//начиная с позиции index***void** Copy(string1 s, **unsigned** index, **unsigned** count, string1 subS);  
  
*//Возвращает текущую длинну строки s***unsigned** Length(string1 s);  
  
*//Возвращает позицию начиная с которой в строке s  
//распологается строка subS***unsigned** Pos(string1 subS, string1 s);  
  
  
  
#endif *//ASD\_2\_STR\_H*

Модуль

#include **"str.h"**#include **<stdio.h>  
  
short** STRING\_ERROR;  
  
  
  
**int** maxN=1024;  
*//Запись данных в строку st из строки s  
//Строка s заканчивается нулевым символом***void** WriteToStr(string1 st, **char** \*s){  
 **int** i = 0;  
 **while** (s[i] != **'\0'**){  
 st->s[i] = s[i];  
 i++;  
 }  
 st->N = i;  
 st->s[i++] = **'\0'**;  
 **if**(i == maxN)  
 STRING\_ERROR = STRING\_INPUT\_ERROR;  
 **else** STRING\_ERROR = STR\_SUCCESSFUL;  
  
}  
  
*//Запись данных в строку s из строки st  
//Строка s заканчивается нулевым символом***void** WriteFromStr(**char** \*s, string1 st){  
 **int** i = 0;  
 **while** (st->s[i] != **'\0'**){  
 s[i] = st->s[i];  
 i++;  
 }  
 st->N = i;  
 s[i++] = **'\0'**;  
  
 STRING\_ERROR = STR\_SUCCESSFUL;  
  
}  
  
*//Ввод строки s с клавиатуры***void** InputStr(string1 st)  
{  
  
 **char** k = getchar();  
  
 **unsigned short** i = 0;  
  
 **while** ((k != **EOF** && k != **'\n'**) && (i < maxN)){  
  
 st->s[i] = k;  
  
 ++i;  
 k = getchar();  
  
 }  
  
 st->N = i;  
  
 st->s[i++] = **'\0'**;  
  
  
 **if**((i == maxN) && (k != **EOF** && k != **'\n'**))  
 STRING\_ERROR = STRING\_INPUT\_ERROR;  
 **else** STRING\_ERROR = STR\_SUCCESSFUL;  
  
}  
*//Вывод строки s на экран монитора***void** OutputStr(string1 st){  
 **int** i = 0;  
 **char** a;  
 **while** (st->N != i){  
 putchar(st->s[i]);  
 i++;  
 }  
 STRING\_ERROR = STR\_SUCCESSFUL;  
}  
  
*//Выполняет конкатенацию строк s1 и s2 результат помещает в sRez***void** Concat(string1 s1, string1 s2, string1 srez){  
 **if**(s1->N+s2->N>srez->N){  
 STRING\_ERROR = STRING\_CONCATEN\_ERROR;  
 }  
 **for** (**int** i = 0; i<s1->N; i++){ *//Скопируем в срез первую строку* srez->s[i] = s1->s[i];  
 }  
 **int** i = s1->N;  
 **for** (**int** j = 0; j<s2->N; j++){ *//Соединим срез со второй строкой* srez->s[j+i] = s2->s[j];  
 }  
 srez ->N = s1->N + s2->N;  
 srez->s[srez->N] = **'\0'**;  
}  
  
  
*//Сравнивает строки s1 и s2 возвращает 0 если  
//s1 == s2; 1 если s1 > s2; -1 если s1 < s2***int** Comp(string1 s1, string1 s2){  
 **int** i = 0;  
 **if** (s1->N > s2->N){  
 STRING\_ERROR = STR\_SUCCESSFUL;  
 **return** 1;  
 }  
 **if** (s2->N > s1->N){  
 STRING\_ERROR = STR\_SUCCESSFUL;  
  
 **return** -1;}  
 **while** ((s1->s[i] == s2->s[i])&&(i < s1->N)){  
 i++;  
 }  
 **if** ((i == s1->N)&&(i == s2->N)) {  
 STRING\_ERROR = STR\_SUCCESSFUL;  
 **return** 0;  
 }  
 **if** (s1->s[i] > s2->s[i]) {  
 STRING\_ERROR = STR\_SUCCESSFUL;  
  
 **return** 1;  
 }*//Первая строка больше, первый из неравных символов имеет больший код* **if** (s1->s[i] < s2->s[i]){  
 STRING\_ERROR = STR\_SUCCESSFUL;  
  
 **return** -1;  
}  
}  
  
  
*//Удаляет count символов из строки s  
//начиная с позиции index***void** Delete(string1 s, **unsigned** Index, **unsigned** Count){  
 **for** (**int** i = Index; i < s->N; i++){  
 s->s[i] = s->s[i+Count]; }  
 s->s[s->N - Count] = **'\0'**;  
 s->N = s->N - Count;  
 STRING\_ERROR = STR\_SUCCESSFUL;  
}  
  
  
*//Вставляет подстроку subS в строку s  
//начиная с позиции index***void** Insert(string1 subS, string1 s, **unsigned** index)  
{  
  
 **if** (index > s->N)  
 STRING\_ERROR = STRING\_NO\_PLACE;  
 **else if**(subS->N + s->N > maxN)  
 STRING\_ERROR = STRING\_INSERT\_ERROR;  
 **else**{  
  
 **unsigned short** i = s->N + 1;  
  
 **while** (i >= index){  
  
 s->s[i + subS->N] = s->s[i];  
 --i;  
 }  
  
 i = index;  
 **unsigned short** j = 0;  
  
 **while** (j < subS->N){  
  
 s->s[i] = subS->s[j];  
 ++i;  
 ++j;  
  
 }  
 s->N=s->N+subS->N;  
  
  
 }  
  
}  
  
  
  
  
  
*//Записывает count символов в строку subS из строки s  
//начиная с позиции index***void** Copy(string1 s, **unsigned** index, **unsigned** count, string1 subS)  
{  
 **if**(index + count > s->N)  
 STRING\_ERROR = STRING\_NO\_PLACE;  
 **else if**(count > maxN)  
 STRING\_ERROR = STRING\_NO\_PLACE;  
 **else**{  
 **unsigned short** i = 0;  
 **unsigned short** rBord = index + count;  
  
 **while** (index < rBord){  
  
 subS->s[i] = s->s[index];  
 ++i;  
 ++index;  
  
 }  
 subS->N = i;  
 STRING\_ERROR = STR\_SUCCESSFUL;  
  
 }  
  
}  
  
*//Возвращает текущую длинну строки s***unsigned** Length(string1 s){  
 **return** s->N;  
}  
*//Возвращает позицию начиная с которой в строке s  
//распологается строка subS или длинну строки если ее нет то 0***unsigned** Pos(string1 subS, string1 s)  
{  
 **unsigned short** j = 0;  
 **unsigned short** lens = s->N;  
 **unsigned short** lensubS = subS->N;  
 **unsigned short** len=lens-lensubS;  
 **if**(s->N<subS->N)  
 STRING\_ERROR = STRING\_Pos\_ERROR;  
 **else** STRING\_ERROR = STR\_SUCCESSFUL;  
 **for** (**unsigned short** i = 0; i <= len; ++i){  
 j = 0;  
 **while** ((j < subS->N) && (s->s[i + j] == subS->s[j]))  
 ++j;  
 **if**(j == subS->N)  
 **return** i+1;  
 }  
 **return** 0;  
}

Основная программа

#include **<stdio.h>**#include **<stdlib.h>**#include **"str.h"  
  
  
  
unsigned** LastPost(string1 s1,string1 s2){  
 **unsigned short** lens1=s1->N;  
 **unsigned short** lens2=s2->N;  
 STRING\_ERROR = STR\_SUCCESSFUL;  
 **unsigned short** n,i=1,q=0;  
 n=Pos(s2,s1);  
 **while**(n!=0&&i<lens1&&lens1>lens2) {  
 q+=n;  
 n=Pos(s2,s1);  
 Delete(s1,0,(n+lens2));  
 i++;  
 }  
 **return** q;  
  
}  
**int** main() {  
 string1 string\_1 , string\_2;  
  
 InputStr(string\_2);  
 printf(**"Error: %d\n"**, STRING\_ERROR);  
 InputStr(string\_1);  
 printf(**"Error: %d\n\n"**, STRING\_ERROR);  
  
 printf(**"LastPost %d\n"**, LastPost(string\_1,string\_2));  
 printf(**"Error: %d\n\n"**, STRING\_ERROR);  
  
 **return** 0;  
}



**Стек**

Заголовочный файл

#ifndef **ASD\_2\_STEK\_H**#define **ASD\_2\_STEK\_H***// размер стека***const short** stacksize = 100;  
*// если операция прошла успешно***const short** stackOk = 0;  
*// если стек пустой***const short** stackEmpty = 1 ;  
*// если стек переполнен***const short** stackfull = 2;  
*// отображение исключающих ситуаций***extern short** stackError;  
  
**typedef int** BaseType;  
  
**typedef struct** {  
 BaseType Buf[100];  
 **short** n;  
}stack;  
  
**typedef** stack \*stack1;  
**void** initStack(stack1\* s);  
  
**void** putStack(stack1\* s, BaseType el);  
  
**void** getStack(stack1\* s, BaseType \* el);  
  
**short** emptyStack(stack1 s);  
  
**short** fullStack(stack1 s);  
#endif *//ASD\_2\_STEK\_H*

Модуль

#include **"stek.h"  
short** stackError;  
*// Назначение: возвращает stackFull, если стек s переполнен, иначе возвращает stackOk***short** fullStack(stack1 s){  
  
 **return** (s->n == stacksize) ? stackfull : stackOk;  
  
}  
  
*// Назначение: возвращает stackEmpty, если стек s пуст, иначе возвращает stackOk***short** emptyStack(stack1 s){  
  
 **return** (!s->n) ? stackEmpty : stackOk;  
  
}  
  
*// Назначение: инициализирует стек s***void** initStack(stack1\* s){  
  
 (\*s)->n = 0;  
  
 **return**;  
}  
  
*// Назначение: включает элемент el в стек s, если стек не переполнен***void** putStack(stack1\* s, BaseType el){  
  
 **if** (!(stackError = fullStack(\*s))){  
  
 (\*s)->Buf[(\*s)->n] = el;  
  
 (\*s)->n += 1;  
  
 }  
 **return**;  
}  
  
*// Назначение: исключает элемент el из стека s, если стек не пуст***void** getStack(stack1\* s, BaseType \* el){  
  
 **if** (!(stackError = emptyStack(\*s))){  
  
 \*el = (\*s)->Buf[(\*s)->n];  
  
 (\*s)->n -= 1;  
  
 }  
 **return**;  
}