

**Методические указания к лабораторной работе № 4 по курсу
ОСНОВЫ ПРОГРАММИРОВАНИЯ
ГУИМЦ**

**"Строки и указатели в СИ"
(6 часов)**

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
1. 1. Цель лабораторной работы № 4 по дисциплине ОП(Основы программирования)	4
2. 2. Порядок выполнения лабораторной работы	4
3. 3. Основные понятия.....	4
3.1. 3.1 Строки в СИ.....	4
3.2. 3.2 Описание строк и инициализация строк	4
3.3. 3.3 Основные операции со строками	6
3.4. 3.4 Ввод и вывод строк	7
3.5. 3.5 Манипуляция со строками и в строке.....	8
3.6. 3.6 Преобразование к нижнему или верхнему регистру	9
3.7. 3.7 Преобразование чисел в строку и обратно	9
3.8. 3.8 Дублирование динамических строк.....	10
3.9. 3.9 Массивы строк и указателей на строки	10
3.10. 3.10 Выделение подстрок на множестве разделителей.....	11
3.11. 3.11 Проверка класса символов.....	12
3.12. 3.12 Сортировка строк	12
3.13. 3.13 Динамические строки.....	13
3.14. 3.14 Функции sprintf и sscanf.....	15
3.15. 3.15 Библиотеки функций и классы для строк.....	15
3.16. 3.16 Аргументы командной строки.....	16
4. 4. Примеры программы с использованием строк.....	17
4.1. 4.1 Примеры, описанные в теоретической части ЛР	17
4.2. 4.2 Пример на выделение подстроки	18
4.3. 4.3 Обмен строк одинакового размера	18
4.4. 4.4 Поиск символа в строке	18
4.5. 4.5 Замена всех символов в строке	19
4.6. 4.6 Поиск вхождения подстроки в строке	19
4.7. 4.7 Формирование действительного числа с точкой и знаком	20
5. 5. Контрольные задание ЛР №4.	20
5.1. 5.1 Создание большой строки ФИО.....	20
5.2. 5.2 Создание строки ФИО инициалами.....	21
5.3. 5.3 Замена символов.....	21
5.4. 5.4 Ввод и вывод массива строк.....	22
5.5. 5.5 Изменение порядка символов в строке	23
5.6. 5.6 Динамические строки.....	24
6. 6. Варианты заданий для студентов СУЦ.	24
7. 7. Дополнительные требования для студентов СУЦ (д.т.).	25
7.1. 7.1 Создание функции SubString	25
7.2. 7.2 Создание функции SwapString для строк равных по длине.....	25
7.3. 7.3 Изменение порядка символов в строке	25
7.4. 7.4 Сортировка массива строк.....	26
7.5. 7.5 Минимум и максимум в массиве строк.....	26
7.6. 7.6 Замена подстроки в строке	26
7.7. 7.7 Функция сравнение строк.....	26
7.8. 7.8 Создание функции DSwapString для динамических строк	26
7.9. 7.9 Изучение библиотечных функций.	26
8. 8. Демонстрация, защита ЛР и отчет по ЛР.....	27
9. 9. Контрольные вопросы по ЛР.	27
10. Литература.....	27

1. 1. Цель лабораторной работы № 4 по дисциплине ОП(Основы программирования)

Целью данной ЛР по дисциплине ОП является получение навыков работы с переменными типа строка на языке программирования СИ. Студенты используют консольные проекты и отлаживают программы в среде программирования MS VS 2005/2008/2010. Студенты знакомятся с основными строковыми операциями, преобразованиями строк, с библиотечными функциями для работы со строками, проверяют работу отлаженных примеров и делают контрольные задания. Они выполняют отладку программы по своему варианту и получают исполнимую программу, готовую к выполнению, оформляют отчет по ЛР и защищают его.

2. 2. Порядок выполнения лабораторной работы

1. Познакомиться с содержанием методических указаний и основными понятиями ЛР (разделы 3 и 4)
2. Проработать порядок выполнения работы (раздел 5).
3. Создать консольные проекты для проверки примеров и выполнения задания ЛР (разделы 3,4).
4. Проверить в данном проекте примеры из методических указаний, выполнив их в отладчике в пошаговом режиме (раздел 4).
5. Написать программу заданий ЛР по варианту, выданному преподавателем и отладить ее (раздел 5).
6. **Продемонстрировать работу программы преподавателю в режиме отладчика по шагам и изменяемыми переменными.**
7. Подготовить отчет по ЛР по представленному шаблону (раздел 8).
8. Защитить ЛР с предоставлением отчета и ответами на контрольные вопросы (разделы 8,9).
9. Для продвинутых студентов выполнить задания для дополнительных (необязательных) требований и также отобразить их в отчете по ЛР (раздел 7).

3. 3. Основные понятия

В теоретической части описания лабораторной работы вводятся основные понятия и рассматриваются принципы для работы со строками на языке программирования СИ.

3.1. 3.1 Строки в СИ

Тип данных строка является одним из самых важных в информационных технологиях. В переменных и массивах данного типа запоминаются данные, характерные для использования в базах данных (названия, фамилии, имена, даты и т.д.). В принципе, любые данные могут храниться в символьном формате, а затем перед использованием преобразовываться в нужный формат с помощью специальных функций языка. К сожалению, этот тип отсутствует в списке стандартных типов языка СИ. В развитии СИ++ такой тип реализован на основе классов и библиотек. Несмотря на это в базовом языке СИ предусмотрены многочисленные возможности, библиотеки и средства для работы со строками. Строка в языке СИ представляется символьным массивом (**char**). Предусмотрены возможности: копирования и слияния строк, их ввода и вывода, выделение части строки, поиск в строке и многие другие возможности. Рассмотрим их.

3.2. 3.2 Описание строк и инициализация строк

Описание строки совпадает с описанием символьного массива. При этом допустимы все изученные вами ранее возможности использования и инициализации массивов. Например:

```
char Str1[10]; // Описана строка максимальной длины 9 символов
```

```

char Str1[20]; // Описана строка максимальной длины 19 символов
Str1[0] = 'П';
Str1[1] = 'р';
Str1[2] = 'и';
Str1[3] = 'м';
Str1[4] = 'е';
Str1[5] = 'р';
Str1[6] = '\0'; // Нулевой символ заносится в этом случае самостоятельно в конце текста

```

Выполним вывод строк для разных случаев:

```

printf("Вывод строки Str(ошибка) = \"%s\\\" \\n", Str); // нет конца строки
Str[0] = '\0'; // Зададим в начале нулевой символ
printf("Вывод строки (пустая) Str = \"%s\\\" \\n", Str);
printf("Вывод строки (текст) Str1 = \"%s\\\" \\n", Str1);

```

Получим (вывод строки без нулевого символа в конце дает ошибку):

```

Вывод строки Str(ошибка) = "MMMMMMMMMMMMMMMM{Cфй@э-"
Вывод строки (пустая) Str = ""
Вывод строки (текст) Str1 = "Пример"
Для продолжения нажмите любую клавишу . . .

```

Однако можно использовать конструкцию инициализации строки, использованную ниже. Отметим, что нулевой символ при такой инициализации заносится автоматически, нужно следить затем, чтобы длина строки инициализации не превышала размер символьного массива:

```

char Str2[20] = "Пример строки RTL!!";
char Str3[20] = {'П', 'р', 'и', 'м', 'е', 'р', '!', '!'}; // Можно и так для каждого элемента – 1 символ
printf("Вывод строки (текст) Str2 = \"%s\\\" \\n", Str2);
printf("Вывод строки (текст) Str3 = \"%s\\\" \\n", Str3);

```

Когда исходный размер массива не так существенен, то можно не указывать первоначальный максимальный размер в квадратных скобках. В этом случае размер массива будет вычислен автоматически с учетом добавления нулевого символа. В этом случае можно записать так:

```

char Str4[] = "Пример строки с размером в длину строки инициализации!"; // Подсчитайте сами (55)

```

Печать:

```

printf("Вывод строки (текст) Str2 = \"%s\\\" \\n", Str2);
printf("Вывод строки (текст) Str3 = \"%s\\\" \\n", Str3);
char Str4[] = "Пример строки с размером в длину строки инициализации!"; // Подсчитайте сами (55)
printf("Вывод строки размером - %d (текст) Str4 = \"%s\\\" \\n", strlen(Str4), Str4);

```

```

Вывод строки (текст) Str2 = "Пример строки RTL!!"
Вывод строки (текст) Str3 = "Пример!"
Вывод строки размером - 55 (текст) Str4 = "Пример строки с размером в длину строки инициализации!"

```

Здесь **strlen()** - функция библиотеки для работы со строками **<string.h>** (см. ниже).

Для заполнения строки можно использовать библиотечную функцию **strcpy()**, которая входит в состав библиотеки RTL. Для этого нужно подключить заголовочный файл этой библиотеки:

```

#include <string.h>

```

...

```

char Str5[30];
printf("Вывод строки размером - %d (текст) Str5 = \"%s\\\" \\n", strlen(Str5), Str5); // размер
//неправильный т.к. нет нуль символа в конце
strcpy(Str5, "Строка заносимая в программе!"); //Заполнение функцией копирования строк
printf("Вывод строки размером - %d (текст) Str5 = \"%s\\\" \\n", strlen(Str5), Str5); //после
//копирования добавляется нуль символ

```

```
Вывод строки размером - 94 (текст) Str5 = "MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM  
ММВывод строки размером - 29 (текст) Str5 = "Строка заносимая в программе!"
```

3.3. 3.3 Основные операции со строками

```
#include <string.h>
...
char Str5[30];
strncpy(Str5, Str4 , 29); // Копироваться более 29 символов не будет!
```

```
char Name[10]; // Описана строка Name максимальной длины 9 символов
char family[20] = "Пример"; // Описана строка family и инициализирована
strcpy(family, "Петров"); // копирование
strcpy(Name, "Иван"); // копирование
strcat (family , " "); // пробел между строками
strcat (family , Name); // сложение строк
printf("Печать строки \"%s\"=\n",family); // Печать
char Fam[14] ;
strcpy_s(Fam , "Петров");
printf ( "Строка Fam = %s имеет длину - %d \n", Fam , strlen (Fam));
strcpy_s(Fam , "Спиридонов");
printf ( "Строка Fam = %s имеет длину - %d \n", Fam , strlen (Fam));
printf ( "Максимальный размер Fam = %s равен - %d \n", Fam , sizeof (Fam)/sizeof (char) );
```

Строка Fam = Петров имеет длину - 6
Строка Fam = Спиридонов имеет длину - 10
Максимальный размер Fam = Спиридонов равен - 14

```
char Person[30];
```

ОП ГУИМЦ 2024 ЛРН№4

```
char Name1[20] = "Сергей";
char Fam1[20] = "Большаков";
// Копирование и слияние без контроля
strcpy(Person, Fam1);
strcat(Person, " "); // Нужно для пробела между фамилией и именем
strcat(Person, Name1);
printf("Фамилия и имя = %s !\n", Person);
```

Результат получим такой:

Большаков Сергей !

Если предусмотреть контроль копирования и слияния строк, то программа будет выглядеть так (используем функцию **strlen** и макрос **sizeof (char)**):

```
strncpy(Person, Fam1, sizeof (Person)/sizeof (char) - 1);
// Следующее нужно для пробела между фамилией и именем
strncat(Person, " ", sizeof (Person)/sizeof (char) - strlen (Person) - 1);
// Допустимо только - sizeof (Person)/sizeof (char) - strlen (Person) - 1
strncat(Person, Name1, sizeof (Person)/sizeof (char) - strlen (Person) - 1);
printf("Фамилия и имя = %s !\n", Person);
```

Результат будет аналогичным, но в данном тексте исключается затирание оперативной памяти в программе, а, следовательно, и ошибки с ним связанные.

3.4. 3.4 Ввод и вывод строк

Для ввода/вывода в языке СИ используются специальные библиотеки. Особенности файлового ввода/вывода будет посвящена отдельная лабораторная работа. Ввод данных в стандартном СИ (в консольном проекте) предполагается с клавиатуры, а вывод на экран дисплея. Возможен ввод/вывод с файловой системой, но мы будем этими технологиями заниматься отдельно. Здесь же мы отметим, что предусмотрено три возможности выполнять ввод/вывод:

- Ввод/вывод верхнего уровня (Потоки, форматирование и буферизация);
- Ввод/вывод среднего уровня (Буферизация, минимальное форматирование);
- Ввод/вывод нижнего уровня (Буферизация и работа с байтами).

К функциям верхнего уровня относятся известные вам функции **printf** и **scanf**, которые для операций ввода вывода используют параметр форматирования “%s”. Кроме этого, для форматирования может указываться максимальная длина ввода вывода “%.10s” (10 символов), минимальная длина ввода вывода “%5s” (5 символов), совместное ограничение “%5.10s” и выравнивание влево “%-5.10s”. Примеры:

```
char Str10[ 20 ] = {"Пример"}; // строки!
printf("Строка -> '%s' \n", Str10);
printf("Строка -> '%10s' \n", Str10);
printf("Строка -> '%-10s' \n", Str10);
printf("Строка -> '%3.20s' \n", Str10);
printf("Строка -> '%-3.4s' \n", Str10);
```

Результат работы операторов вывода на экран будет следующий:

```
Строка -> 'Пример'
Строка -> '      Пример'
Строка -> 'Пример      '
Строка -> 'Пример'
Строка -> 'Прим'
```

Для функции **scanf** также указывается формат вида - “%s” (без ограничения длины вводимой строки) и с ограничением в заданное число символов - “%5s”.

```
scanf ("%s", Str10);
printf("Строка (scanf1) -> '%s' \n", Str10);
scanf ("%5s", Str10);
printf("Строка (scanf2) -> '%s' \n", Str10);
```

Получим результат:

```
1234567890
```

```
Строка (scanf1) -> '1234567890'
```

```
1234567890
```

```
Строка (scanf2) -> '12345'
```

При другом вводе (с пробелом!) результат будет другим, так как пробел ограничивает размер вводимой строки для одной функции **scanf**:

```
12345_678909
```

```
Строка (scanf1) -> '12345'
```

```
Строка (scanf2) -> '67890'
```

Для ввода с пробелами и другими особенностями для строк используют функции **gets** и **puts** и другие. Вам предлагается здесь познакомиться с этими функциями самостоятельно. Мы рассмотрим данные функции позже.

3.5. 3.5 Манипуляция со строками и в строке

Для сравнения строк в СИ используются библиотечные функции: **strcmp**, **strncmp**, **_strnicmp** и другие. Пример и результаты приведены ниже.

```
// strcmp , strncmp - сравнение строк
char Name[24] = "Василий"; // Посмотреть в отладчике нуль-терм.
setlocale( LC_ALL, "" ); // нужно подключить locale.h
//
if ( strcmp(Name , "Василий") == 0) printf ("Строки равны (идентичны)! \n" );
if ( strcmp(Name , "Алексей") > 0) printf ("Строки не равны (1-я > 2- й)! \n" );
if ( strcmp(Name , "Федор") < 0) printf ("Строки не равны (1-я < 2- й)! \n" );
// число сравниваемых данных - 5
if ( strncmp(Name , "Василиса" , 5 ) == 0) printf ("Строки равны (идентичны)! \n" );
// нет ограничения
if ( strcmp(Name , "Василиса" ) == 0)
    printf ("Строки равны (идентичны)! \n" );
else
    printf ("Строки не равны ! \n" );
// _strnicmp – сравнение без учета регистра
if ( _strnicmp("Name" , "NAME" , 3 ) == 0) printf ("Строки равны (_strnicmp - идентичны)! \n" );
```

Результат получим такой:

```
Строки равны (идентичны)!
Строки не равны (1-я > 2- й)!
Строки не равны (1-я < 2- й)!
Строки равны (идентичны)!
Строки не равны !
Строки равны (_strnicmp - идентичны)!
```

Для поиска первого и последнего вхождения символа в строке используются функции: **strchr** и **strrchr**. Пример их применения дан ниже, а функции **strcspn** и **strspn** ищут вхождение или не вхождение подстроки в строку – номер совпадений.

```
// strchr - первое вхождение "и" в слове Василий
char Name[24] = "Василий"; // Посмотреть в отладчике нуль-терм.
printf ("Строка исходная = %s   Часть с первым найденным \"и\" = %s \n" , Name , strchr(Name , 'и' ));
// strrchr - последнее вхождение "и" в слове Василий

printf ("Строка исходная = %s   Часть с последним найденным \"и\" = %s \n" , Name , strrchr(Name , 'и' ));
int num = strspn(Name5, "Ba");    // num = 2
num = strcspn(Name5, "ил");    // num = 3
```

Результат получим такой:

Строка исходная = Василий Часть с первым найденным "и" = илий
 Строка исходная = Василий Часть с последним найденным "и" = ий

Контроль строки на содержания подмножества символов производится функцией **strspn**.

```
// strspn
char string1[] = "cabbage";
int result;
result = strspn( string1, "abc" );
printf( "Размер подстроки '%s', в которой только символы: a, b, or c "
        "= %d байт\n", string1, result );
```

Результат получим такой:

Размер подстроки 'cabbage', в которой только символы: a, b, or c = 5 байт

3.6. 3.6 Преобразование к нижнему или верхнему регистру

Для преобразования строки к нижнему и верхнему регистру (строчные и прописные буквы) применяют функции **strlwr** и **strupr**, соответственно.

```
// ПРЕОБРАЗОВАНИЕ К НИЖНЕМУ И ВЕРХНЕМУ РЕГИСТРУ
char string5[10] = "";
strcpy( string5, "Sample");
printf( " %s\n", _strupr ( string5 ) );
setlocale( LC_ALL, "" );
strcpy( string5, "Пример!"); // русские не преобразовывает без setlocale( LC_ALL, "" )
printf( " %s\n", strupr ( string5 ) );
printf( " %s\n", strlwr ( string5 ) );
```

Результат получим такой:

SAMPLE
 ПРИМЕР!
 пример!

3.7. 3.7 Преобразование чисел в строку и обратно

Очень важные действия в языках программирования связаны с взаимными преобразованиями данных. Например, число преобразуется в строку или наоборот строка с цифровым текстом преобразуется в число. Для таких преобразований в СИ есть функции, которые включены в библиотеку **stdlib.h**. Ниже показаны примеры такого использования.

Для преобразования в строку чисел и обратно используются следующие функции:

- **atoi** -преобразование строки в число типа **int** (целое)
- **itoa** - преобразование числа типа **int** в строку
- **atof** - преобразование строки, в представляемое ей число типа **float**
- **gcvt**- преобразование числа типа **double** в строку

Примеры использования функций преобразования чисел:

```
// Преобразования чисел в строку
int iVar = atoi("125");// atoi и itoa
printf( "Строку в целое 125 => %d\n", atoi("125") );
char sVar[20]="";
printf( "Целое в строку 5 => %s\n", itoa ( 15 , sVar , 10 ) );
double dVar = atof("10,5") ; // atof
printf( "Строку в вещественное 10.5 => %10.2f1\n", dVar );
dVar = 15.7; // atof и gcvt
printf( "вещественное в строку 15.7 => %s \n", _gcvt(dVar,5,sVar) ); // gcvt
```

В результате получим:

Строку в целое 125 => 125

Еще примеры использования функций преобразования чисел:

Целое в строку 5 => 15

Строку в вещественное 10.5 => 10,50

вещественное в строку 15.7 => 15,7

// Преобразование из строки в число и чисел в строку

```
char Buf[15];
int Dec , Sign;
//
printf("Из Строки (printf) -> '%s' в целое - %d\n", "125" , atoi( "125" ) );
printf("Из Строки (printf) -> '%s' в вещественное - %8.3f или %e \n", "125.5" , atof( "125.5"
));
printf("Из целого (printf) -> '%d' в строку - %s в строку(itoa_s) - %s \n", 125 , itoa( 125, Buf
, 10 ), _itoa_s( 25, Buf , 14 ,10 ) );
printf("Из целого с защитой Buf (printf) -> '%s' \n", Buf );
_gcv( -35.5, 12, Buf );
printf("Из вещественного Buf (_gcv) -> '%s' \n", Buf );
//
```

Результат получим такой:

Из Строки (printf) -> '125' в целое - 125

Из Строки (printf) -> '125.5' в вещественное - 125.500 или 1.299551e-257

Из целого (printf) -> '125' в строку - 125 в строку(itoa_s) - (null)

Из целого с защитой Buf (printf) -> '125'

Из вещественного Buf (_gcv) -> '-35.5'

3.8. 3.8 Дублирование динамических строк

Для дублирования строки с выделением памяти применяют функцию **strdup** (или **_strdup**). После ее использования ее освобождают функцией **free**. Отметим что эта операция отличается от операции копирования указателей.

```
char buffer[] = "Это текст буферной строки!";
char *newstring;
printf( "Исходная строка 1: %s\n", buffer );
newstring = strdup( buffer ); // Дублирование строки динамически выделяется память
newstring[2] = '*'; // Изменение 3-го символа в строке
printf( "Копия строки: %s\n", newstring );
printf( "Исходная строка 2: %s\n", buffer );
free( newstring ); // Изменение освобождение памяти под дубль строку
```

Результат получим такой:

Исходная строка 1: Это текст буферной строки!

Копия строки: Эт* текст буферной строки!

Исходная строка 2: Это текст буферной строки!!

3.9. 3.9 Массивы строк и указателей на строки

Массив строк – это двумерный массив типа char, в котором в каждой строке должен присутствовать нулевой символ ('\0'). Если такой массив инициализируется, то нулевой символ обеспечивается компилятором:

```
char MasStr[5][15]={ "Строка0", "Строка1", "Строка2", "Строка3", "Строка4"};
```

При работе с таким массивом индексное выражение MasStr[i] является строкой.

```
// Печать массива строк
for (int i = 0; i < 5 ; i++ )
printf ( "%s\n",MasStr[i]);
```

Массив указателей на строки – это массив указателей на тип char (инициализируем его):

```
char str1[10] = "Str1";
char str2[10] = "Str2";
char str3[10] = "Str3";
```

ОП ГУИМЦ 2024 ЛРН№4

```
char * MasPtrStr[3]={str1,str2,str3}; // Инициализация строками
// Печать массива указателей на строки
for (int i = 0; i < 3 ; i++ )
printf ("%s\n",MasPtrStr[i]);
```

Получим в результате:

```
Строка 0
Строка 1
Строка 2
Строка 3
Строка 4
Str1
Str2
Str3
```

Указатели на строки описывается как обычный указатель на char:

```
strcpy(family, "Петров"); // копирование
strcpy(Name, "Иван"); // копирование
strcat (family , Name); // сложение строк
```

```
char * PtrStr = family; // инициализация
printf("Печать строки по указателю(family) \"%s\" \n",PtrStr); // Печать
PtrStr = Name;
printf("Печать строки по указателю(Name) \"%s\" \n",PtrStr); // Печать
```

Результат:

```
Печать строки по указателю(family) "Петров Иван"
Печать строки по указателю(Name) "Иван"
```

3.10. 3.10 Выделение подстрок на множестве разделителей

Удобной возможностью разбиения строки на части является применения функции **strtok**. На основе заданного множества разделителей (у нас в примере: пробел, запятая, табуляция и конец строки) последовательно в цикле выделяются подстроки. Такие действия, соответствуют грамматическому разбору строки и часто называются парзингом (parse). В примере подстроки выводятся на экран.

```
/// ВЫДЕЛЕНИЕ ПОДСТРОК НА МНОЖЕСТВЕ РАЗДЕЛИТЕЛЕЙ
char strText4[] = "Строка\tdля ,,разделения на tokens\n по множеству разделителей\n";
char seps[] = " ,\t\n"; // Допустимые разделители: (пробел, запятая, табуляция и //
конец строки)
char *token; // Указатель на массив подстрок
printf( "Исходная строка:%s\n", strText4 ); // Разделители не видны
printf( "Подстроки (tokens):\n" );
// Получение первой подстроки:
token = strtok(strText4, seps ); //
while( token != NULL ) // проверка наличия новых подстрок
{ // Вывод подстрок в цикле
printf( " %s\n", token );
// Новая подстрока :
token = strtok( NULL , seps ); } //
```

Результат получим такой:

```
Исходная строка:Строка для ,,разделения на tokens
по множеству разделителей
Подстроки (tokens):
Строка
для
```

разделения
на
tokens
по
множеству
разделителей

Для большей понятности уменьшим число разделителей (Оставим только запятую – ","):

```
char strText4[] = "красный,зеленый,желтый\n";
char seps[] = ","; // Допустимый разделитель: (запятая
```

Результат получим такой:

Исходная строка:красный,зеленый,желтый

Подстроки (tokens):

красный
зеленый
желтый

3.11. 3.11 Проверка класса символов

Функции, используемые для классификации символов, приведены ниже (библиотека

<ctype.h>):

- **isalnum(c)** **isalpha(c)** или **isdigit(c)** есть истина
- **isalpha(c)** **isupper(c)** или **islower(c)** есть истина
- **isctrl(c)** управляющий символ
- **isdigit(c)** десятичная цифра
- **isgraph(c)** печатаемый символ кроме пробела
- **islower(c)** буква нижнего регистра
- **isprint(c)** печатаемый символ, включая пробел
- **ispunct(c)** печатаемый символ кроме пробела, буквы или цифры
- **isspace(c)** пробел, смена страницы, новая строка, возврат каретки, табуляция, вертикальная табуляция
- **isupper(c)** буква верхнего регистра
- **isxdigit(c)** шестнадцатеричная цифра

Пример использования функций классификации символов:

// Проверка Класса символа

Пример:

```
if (isdigit( '5'))      printf( "Символ 5 -> цифра\n");
if (isalpha( 'm'))      printf( "Символ m -> буква\n");
```

Результат:

Символ 5 -> цифра
Символ m -> буква

3.12. 3.12 Сортировка строк

Сортировка массива строк массива строк по убыванию выполняется аналогично сортировке целого массива, только отличается фрагмент сравнения и обмена элементов в массиве строк. В примере предполагается, что все строки занимают размер не более 9-ти символов. Для обмена используется специальная функция SwapStr, которая предварительно описывается.

```
// Функция обмена строк вне main
void SwapStr (char * S1 , char * S2 )
{
    char TempStr[20];
    strcpy(TempStr , S1 );
```

ОП ГУИМЦ 2024 ЛР№4

```

        strcpy(S1 , S2);
        strcpy( S2 , TempStr);
    };
    //
    #define RazmMas 5
    ...
    // Массив строк инициализируется в программе фамилиями
    char StrMas[RazmMas][10]={"Сидоров", "Алетров", "Иванов", "Жучков", "Акулов"};
    // печать массива строк до сортировки
    printf ("До сортировки \n");
    for (int i =0 ; i < RazmMas ; i++)
        printf (" %d. - %s \n" , i + 1 , &StrMas[i][0]);
    // Сортировка
    for (int k = 0 ; k < RazmMas - 1 ; k++)
        for ( int i =0 ; i < RazmMas - 1; i++)
            { if ( strcmp(&StrMas[i][0] , &StrMas[i +1][0]) > 0 ) // Для убывания по алфавиту
                // Обмен если условие сортировки не соблюдается
                // SwapStr(&StrMas[i][0] , &StrMas[i +1][0]); // Это для ЛР №5
                // Swap без функции
                {
                    strcpy(TempStr , &StrMas[i][0] );
                    strcpy(&StrMas[i][0] , &StrMas[i+1][0]);
                    strcpy( &StrMas[i+1][0] , TempStr);  };
            };
    // печать массива строк после сортировки
    printf ("После сортировки \n");
    for (int i =0 ; i < RazmMas ; i++)
        printf (" %d. - %s \n" , i + 1 , &StrMas[i][0]);

```

Результат получим такой:

До сортировки

1. - Сидоров
2. - Алетров
3. - Иванов
4. - Жучков
5. - Акулов

После сортировки

1. - Акулов
2. - Алетров
3. - Жучков
4. - Иванов
5. - Сидоров

3.13. 3.13 Динамические строки

Часто все переменные и массивы располагаются в оперативной памяти заранее, до начала выполнения программы. Все рассмотренные ранее примеры включали такие переменные и массивы. Размер выделенной памяти под массив/строку, то есть его размерность в этом случае изменить в программе стандартным способом нельзя. Поэтому приходится заранее рассчитывать максимально возможную размерность массива, которая приемлема для всех случаев. Это приводит к перерасходу памяти для отдельных случаев. Для экономии памяти и для построения более универсальных программ используют динамически выделяемую память под массивы, строки и переменные. Такие данные называются динамическими. Важность динамического выделения памяти для строк трудно переоценить. Так как, например, для записи в новую строку текста превосходящего объема, нужно выделить новый больший объем ОП. Это выделение выполняется через предварительное освобождение старого фрагмента динамической памяти.

ОП ГУИМЦ 2024 ЛРН№4

Для строк имя символьного массива рассматривается системой как указатель на **char** (**char ***). Это позволяет его использовать отдельно в функциях и операциях как специальный объект (строка).

В языке СИ оперативная память выделялась с помощью специальных функций (alloc, calloc, malloc, free и т.д.). Они включены в библиотеку СИ – malloc.h. С помощью специальных операций в zpsrt C++ (**new** и **delete**) аналогично можно выделять память во время выполнения программы. Динамические переменные располагаются в оперативной памяти в специальной области. Для работы с такими данными используют указатели и ссылки. Примеры выделения динамической памяти для указателей и массивов строк даны ниже.:

```
#include <malloc.h> // в начало
...
// Динамическая память для строк
char * pStr = (char *) malloc( 10); // malloc
strcpy( pStr , "Динамика!" );
printf ("Динамическая память - %s \n" , pStr);
// Освобождение памяти
free (pStr);
pStr = (char *) calloc (strlen("Новая память!") + 1 , sizeof(char)); // calloc
strcpy( pStr , "Новая память!" );
printf ("Новая память - %s Длиной => %d \n" , pStr, strlen ("Новая память!"));
free (pStr); // Освобождение памяти
```

Получим:

Динамическая память - Динамика!

Новая память - Новая память! Длиной => 13

```
//
int Razm;
//////////
// Динамический массив строк
//////////
printf ("Введите размер массива строк: \n" );
scanf ("%d", &Razm);
char * pStrMas =(char *) calloc (Razm , sizeof(char) * 20); // 20 одна строка
printf ("\nВведите строки массива [%d]: \n" , Razm );
for (int i =0 ; i < Razm ; i++)
    // scanf ("%s", pStrMas + i * 20);
    scanf ("%s", &pStrMas[ i * 20]);
// Распечатка
for (int i =0 ; i < Razm ; i++)
    printf ("Строки массива %d - %s\n", i , &pStrMas[ i * 20] );
//////////
// Слияние массива в отдельную строку
int nMasSize = 200;
char * pBigString = (char *) malloc( nMasSize + 2 + Razm ); // Пробелы воск. знак и ноль
pBigString[0] = '\0';
for (int i =0 ; i < Razm ; i++)
{ strcat( pBigString , &pStrMas[ i * 20]);
  strcat( pBigString , " ");
};
strcat( pBigString , "!");
printf ("Большая строка - %s \n" , pBigString);
free (pBigString);
```

Результат получим такой:

Введите размер массива строк:

3

ОП ГУИМЦ 2024 ЛРН№4

Введите строки массива [3]:

Студенты

изучают

СИ

Строки массива 0 - Студенты

Строки массива 1 - изучают

Строки массива 2 - СИ

Большая строка из введенного массива- Студенты изучают СИ !

3.14. 3.14 Функции sprintf и sscanf

Для работы со строками можно использовать специальные функции форматированного ввода в строку (sprintf) и вывода из строки (sscanf):

Пример:

```
char Sprint[50]; // Строка для вывода (куда выводим)
char sFam[]="Петров";
double dPerem = 55.5;
sprintf(Sprint, "Строка:sFam=%s_dPerem=%f1", sFam, dPerem); // Вывод в строку
printf("Строка Sprint после sprintf: %s\n", Sprint);
char SpInpt[50]; // Строка для ввода (куда вводим)
sscanf(Sprint, "%s", SpInpt); // ввод из строки
printf("Строка SpInpt введенная sscanf из строки Sprint: %s\n", SpInpt);
```

Результат:

Строка Sprint после sprintf: Строка:sFam=Петров_dPerem=55,5000001

Строка SpInpt введенная sscanf из строки Sprint: Стрка:sFam=Петров_dPerem=55,5000001

3.15. 3.15 Библиотеки функций и классы для строк

В системах программирования предусматривается много библиотек для функций различного назначения (например, для работы со строками, выполнения ввода и вывода, работы с массивами и т.д.). Эти библиотеки подключаются с помощью заголовочных файлов или пространств описаний (пространств имен в C++ - **namespace**). Кроме заголовочных файлов для использования библиотек подключаются специальные модули (иногда они подключаются автоматически), содержащие описания функций (*.lib или *.dll). Пример подключения библиотек ввода/вывода и библиотек для работы с математическими и системными функциями:

```
#include <math.h> // Математическая библиотека функций
#include <process.h> // Системная библиотека функций
#include <string.h> // Библиотека функций для работы со строками
#include <stdlib.h> // Стандартная библиотека разных функций
#include <locale.h> // библиотека локализации программ
#include <malloc.h> // библиотека динамической памяти
```

Стандартных библиотек и классов для описания строк очень много. Нужно хорошо знать их назначение и их состав для использования в программах. Чем лучше знания о библиотеках, тем быстрее и безошибочно можно создать сложную программу. В современных системах программирования доступны (большом количестве) библиотеки классов, которые описывают новые дополнительные типы данных.

3.16. 3.16 Аргументы командной строки

При запуске программы в можно задавать параметры (аргументы) командной строки. Их общее число не должно превышать 128 байт.

Параметры программы в режиме отладки задаются в характеристиках проекта: **ПРОЕКТ->Свойства проекта->Свойства конфигурации ->Отладка ->Аргументы Команды:**

Отдельные параметры разделяются пробелами. ОС сама, перед вызовом программы устанавливает нужные значения количества и значений массивов указателей (argc, argv).

Для доступа к параметрам главная функция (**main**) содержит следующий заголовок:

```
void main(int argc, char * argv[])
```

где: **argc** – целая переменная задающая число параметров (нулевой параметр всегда имя программы), а

argv- массив указателей на строки параметров.

Распечатать список параметров можно так:

```
// Параметры программы
printf("Параметры: число = %d\n", argc);
int i;
for (i = 0; i < argc; i++)
printf("%s\n", argv[i]);
```

Если мы зададим параметры так "aaa 333" (см. выше), то получим при распечатке (первый параметр имя программы и путь к ней в ОС):

```
D:\LR4_OP\Debug\LR4_OP_0.exe
aaa
333
```

При необходимости у главной функции может быть задан и третий параметр, который содержит массив указателей на переменные окружения ОС (**envvar**):

```
void main(int argc, char * argv[], char * envvar[])
```

Переменные окружения зависят от компьютера (точнее от установленных программных продуктов на нем) и могут быть распечатаны так:

```
// Переменные окружения
printf("Переменные окружения: \n");
for (i = 0; envvar[i] != NULL; i++)
printf("%d. %s\n", i, envvar[i]);
printf("\n");
```

При распечатке можем получить:

```
Переменные окружения:
0. ALLUSERSPROFILE=C:\ProgramData
1. AMDAPPSDKROOT=c:\Program Files (x86)\AMD APP\
2. APPDATA=C:\Users\User\AppData\Roaming
...
```

3.17. 3.17. Болванка для главного модуля с математической библиотекой

Нужно создать пустой проект в MS VS, как описано выше (раздел 3.4 и 3.5), скопировать через буфер обмена в него текст данного примера прямо из текста данного документа, отладить его (Раздел 3.11), русифицировать (п.3.5) консольное окно и выполнить пошагово. Выбрать один из способов ветвления в программе (переходы или переключатель). Ненужное можно удалить.

```
#define _USE_MATH_DEFINES
```

```
#pragma warning(disable : 4996)
```

```
// Подключение библиотеки математических функций
#include <math.h>
#include <stdio.h>
#include <process.h>
void main(void)
{
    // // п.
    system(" chcp 1251 > nul");
    // Здесь расположить тексты примеров и операторов вычислений из МУ
    // Пример печати
    printf("Главный модуль простого примера с математикой!\n");
    system(" PAUSE");
}
```

4. 4. Примеры программы с использованием строк

Вторая часть задания, помимо первой связанной с изучением теоретического раздела заключается в том, чтобы испытать в проекте СИ уже отлаженные программы и фрагменты программ. Возможно, что, осваивая теоретическую часть работы, вы уже на компьютере проверили выполнение фрагментов текста и применения различных операторов ветвления (из раздела 3), тогда вам будет проще продемонстрировать их работу преподавателю. В дополнение к примерам, расположенным выше нужно испытать и изучить примеры расположенные ниже. Эти действия нужно сделать в отладчике.

Для этого нужно создать пустой проект в MS VS (Test_LR2), как описано выше, скопировать через буфер обмена в него текст данных примеров, отладить его и выполнить.

4.1. 4.1 Примеры, описанные в теоретической части ЛР

Нужно внимательно изучить и проверить работу всех примеров из теоретической части ЛР. Эти примеры расположены выше. Все примеры можно скопировать в свой проект. Все эти задания выполняются обязательно, они не требуют дополнительной отладки и легко (через буфер обмена -Clipboard) переносятся в программу. Все фрагменты должны демонстрироваться преподавателю. В частности, в первой части, там представлены следующие примеры:

1. Описания строк и их инициализации, заполнения строки вручную,
2. Копирования и слияния строк, ввода и вывода строк,
3. Сравнения строк, контроль символов в строке,
4. Преобразование к нижнему или верхнему регистру,
5. Дублирования строк,
6. Выделения составляющих строки по разделителям,
7. Преобразование данных в строку и обратно,
8. Сортировки строкового массива,
9. Использования динамических строк.

Кроме этого ниже представлены примеры, которые могут быть полезными, в том числе и при выполнении контрольных заданий. Их тоже нужно изучить и проверить.

4.2. 4.2 Пример на выделение подстроки

Пример выделения подстроки для строки strText5, начиная с символа с номером **begChar**. Число символов которые будут взяты равно – **sizeSubstr**. Эти значения задаются в программе. Для копирования используется функция - **strncpy**. Для подстроки выделяется динамическая память, размером, на всякий случай, превышающая размер исходной строки на число символов в подстроке. Полученное значение распечатывается.

Выделение подстроки

```
int begChar =4 ; // Начальный символ подстроки
int sizeSubstr = 9; // длина подстроки
char strText5[] = "10123456789 123456789_123456789";
char * pSubstr = (char *) malloc ( sizeof (strText5) + sizeSubstr +1); // Выделение с запасом
strncpy( pSubstr , strText5 + begChar, sizeSubstr + 1 );
pSubstr[5] = '\0'; // Сознательное укорачивание подстроки до 5-ти символов
printf( "Строка: %s Подстрока = %s с =%d число=%d\n",strText5, pSubstr ,begChar,sizeSubstr );
```

Результат получим такой:

Строка: 10123456789 123456789_123456789 Подстрока = 34567 с =4 число=7

4.3. 4.3 Обмен строк одинакового размера

Для обмена строк одинакового размера можно использовать следующий фрагмент. Максимальные размеры данных строк должны быть идентичны (или размеры символьных массивов или размеры выделенной динамической памяти).

```
char TempStr[40];
char S1[40]="Первая строка";
char S2[40]="Вторая строка";
printf( "Перед обменом: %s - %s\n", S1 , S2 );
strcpy(TempStr , S1 ); // Swap
strcpy(S1 , S2);
strcpy( S2 , TempStr);
printf( "После обмена: %s - %s\n", S1 , S2 );
```

Результат получим такой:

Перед обменом: Первая строка - Вторая строка

После обмена: Вторая строка - Первая строка

4.4. 4.4 Поиск символа в строке

Поиск символа, содержащегося в строке на заданном множестве ("abc") символов и определение его номера. Если символов нет, то номер – конец строки, если строка пуста, то нуль.

```
// ...
void test( const char * str, const char * strCharSet )
{
    int pos = strcspn( str, strCharSet );
    printf( "strcspn( \"%s\", \"%s\" ) = %d\n", str, strCharSet, pos );
};
// ...
char string7[] = "cabbage";
// ...
result = strspn( string7, "abc" );
printf( "The portion of '%s' containing only a, b, or c "
        "is %d bytes long у строки - %d\n", string7, result , strlen(string7) );
// strcspn
test( "xyzbxz", "abc" );
```

ОП ГУИМЦ 2024 ЛРН№4

```
test( "xyzbxz", "xyz" );
test( "xyzbxz", "no match" );
test( "xyzbxz", "" );
test( "", "abc" );
test( "", "" );
```

Результат получим такой:

```
strcspn( "xyzbxz", "abc" ) = 3
strcspn( "xyzbxz", "xyz" ) = 0
strcspn( "xyzbxz", "no match" ) = 6
strcspn( "xyzbxz", "" ) = 6
strcspn( "", "abc" ) = 0
strcspn( "", "" ) = 0
```

4.5. 4.5 Замена всех символов в строке

Для заполнения все строки одним символом используется функция `strset`

```
char string7[] = "cabbage";

printf( "Before: %s\n", string7 );
strset( string7, '*' ); //
//
printf( "After: %s\n", string7 );
```

Результат получим такой:

```
Before: cabbage
After: ******
```

4.6. 4.6 Поиск вхождения подстроки в строке

Для поиска в строке (**string**) вхождения подстроки (**str**) используется функция **strstr**, пример ее использования и результаты показаны ниже. В строке мы ищем подстроку “вход”.

```
char str[] = "вход";
char string[] = "Счастливая собака остановилась у входа и залаяла";
char fmt1[] = "      1      2      3      4      5"; // Для разметки полей
char fmt2[] = "12345678901234567890123456789012345678901234567890" ; // Для разметки
char *pdest;
int result1;
printf( "Строка, в которой ищем:\n%s\nПодстроку: %s \n", string, str );
printf( "%s\n%s\n\n", fmt1, fmt2 );
pdest = strstr( string, str );
result1 = (int)(pdest - string + 1);
if ( pdest != NULL )
    printf( "%s Подстрока найдена в позиции %d\n", str, result1 );
else
    printf( "%s Не найдена подстрока!\n", str );
```

Результат получим такой:

```
Строка, в которой ищем:
Счастливая собака остановилась у входа и залаяла
Подстроку: вход
      1      2      3      4      5
12345678901234567890123456789012345678901234567890
вход Подстрока найдена в позиции 34
```

4.7. 4.7 Формирование действительного числа с точкой и знаком

Использование различных функций и приемов для работы со строками продемонстрируем на примере использования функции `_fcvt`, после использования которой полученную строку нужно преобразовать: поставить точку в нужное место и добавить знак.

```
#include <stdlib.h>

...
int Dec;
int Sign;
printf("Из вещественного (printf) -> '%f' в строку - %s \n", 125.5, _fcvt( 125.5, 4, &Dec, &Sign ));
printf(" Параметры(1): где точка - %d и знак %d \n", Dec, Sign);
printf("Из вещественного (printf) -> '%f' в строку - %s \n", -25.5, _fcvt( -25.5, 4, &Dec, &Sign ));
printf(" Параметры(2): где точка - %d и знак %d \n", Dec, Sign);
// Составим число
char RealBuf[15];
strcpy_s( Buf, 14, _fcvt( -25.5, 4, &Dec, &Sign ));
printf("Из вещественного с защитой Buf (printf) -> '%s' \n", Buf );
if ( Sign == 1 )
strcpy(RealBuf, "-");
strncat(RealBuf, Buf, Dec );
RealBuf[Dec + 1] = '\0';
printf("Из вещественного с защитой RealBuf (printf) -> '%s' \n", RealBuf );
strcat (RealBuf, ".");
printf("Из вещественного с защитой RealBuf (printf) -> '%s' \n", RealBuf );
strncat (RealBuf, Buf + Dec, sizeof (Buf + Dec) );
printf("Из вещественного с защитой RealBuf (printf) -> '%s' \n", RealBuf );
```

Результат получим такой:

```
Из вещественного (printf) -> '125.500000' в строку - 1255000
Параметры(1): где точка - 3 и знак 0
Из вещественного (printf) -> '-25.500000' в строку - 255000
Параметры(2): где точка - 2 и знак 1
Из вещественного с защитой Buf (printf) -> '255000'
Из вещественного с защитой RealBuf (printf) -> '-25'
Из вещественного с защитой RealBuf (printf) -> '-25.'
Из вещественного с защитой RealBuf (printf) -> '-25.5000'
```

Нужно создать пустой проект в **MS VS**, как описано выше, скопировать через буфер обмена в него текст данного примера, отладить его и выполнить.

5. 5. Контрольные задание ЛР №4.

5.1. 5.1 Создание большой строки ФИО

Из трех строк, описанных и заранее инициализированных (**Name[14]**, **Fam[20]**, **Otch[20]**), создать новую строку **FIО[56]**, данные разделяются пробелами. Массивы символов для строк инициализировать собственными данными студента (ФИО). Строку вывести на печать.

```
#include <string.h>
//Из трех строк, описанных и заранее инициализированных (Name[14], Fam[20], Otch[20]), со-
здать новую строку FIO
char Fam2 [14] ="Большаков";
char Name2[20] ="Сергей";
char Otch2[20]="Алексеевич";
char FIO[56];
// Копирование и слияние без контроля
strcpy(FIO, Fam2);
strcat(FIO, " "); // Нужно для пробела между фамилией и именем
strcat(FIO, Name2);
strcat(FIO, " "); // Нужно для пробела между и именем и отчеством
```

ОП ГУИМЦ 2024 ЛРН№4

```
strcat(FIO, Otch2);
printf ("Фамилия имя и отчество = %s !\n", FIO);
Получим Результат:
Фамилия имя и отчество = Большаков Сергей Алексеевич !
```

5.2. 5.2 Создание строки ФИО инициалами

Из трех строк, описанных и введенных заранее с клавиатуры (**Name**[14], **Fam**[20], **Otch**[20]), создать новую строку **FIO**[26], в которой имя и отчество задаются инициалами (первыми буквами) с точками (Например, - **Большаков С.А.**). В символьные массивы – строки инициализировать в программе собственными данными студента (ФИО).

```
strcpy(FIO, Fam2);
strcat(FIO, " "); // Нужно для пробела между фамилией и инициалами
strncat(FIO, Name2, 1);
strcat(FIO, ".");
strncat(FIO, Otch2, 1);
strcat(FIO, ".");
printf ("Фамилия имя и отчество = %s !\n", FIO);
////////////////////////////////////
system(" PAUSE");
//
} ;
```

Получим Результат:

Фамилия имя и отчество = Большаков С.А. !

5.3. 5.3 Замена символов

В одномерном символьном массиве (**CharMas**[40]) все символы из перечня (см. таблицу по своему варианту) заменить на один другой символ, заданный вариантом (см. ниже). Массив предварительно инициализировать в программе, самостоятельно придумав его содержание. Для ветвления в цикле при замене символов использовать переключатель (**switch**). Оформить блок-схему программы на MS Visio, как вставленный объект. Массив распечатать до и после замены.

Подсчитать и распечатать число замен (**Counter**).

```
// 5.3. Замена символов \ "иЗлмн№1Лк\ " на пробел
char charMas[] = "Лимон лежал на столе № 135";
int RazmS = strlen(charMas);
printf ("Исходная строка =\n%s !\n", charMas);
// Цикл замен
for (int i=0 ; i < RazmS ; i++)
{
    switch ( charMas[i] )
{case 'и':
case 'З':
case 'л':
case 'м':
case 'н':
case '№':
case '1':
case 'Л':
case 'к':
charMas[i] = ' ';
```

```

Counter++;
    break;    };    };
printf ("Строка после замены символов \"иЗлмн№1Лк\" на пробел\n%s !\n", charMas);
printf ("=\n%s !\n", charMas);
printf ("Число замен=\n%d \n", Counter);

```

Получим результат:

Исходная строка =

Лимон лежал на столе № 135 !

Строка после замены символов "иЗлмн№1Лк" на пробел

о ежа а сто е 5 !

Число замен= 11

5.4.5.4 Ввод и вывод массива строк

Написать фрагмент программы для ввода с клавиатуры массива строк (**MasStr** – минимум пять строк). Число вводимых строк предварительно вводится с клавиатуры. Распечатать введенный массив строк в столбик с указанием номера каждой строки. Порядок вывода строк (прямой или обратный) определяется вариантом (см. ниже). При выводе массива на экран оформить рамку символом точка (“.”).

```

////////////////////////
// Динамический массив строк
////////////////////////
#include <malloc.h>

...
// 5.4 Ввод массива строк
int MasSize;
printf ("Введите размер массива строк: \n" );
scanf ("%d", &MasSize);
char * pStrArray =(char *) calloc (MasSize , sizeof(char) * 20); // 20 символов - одна строка
printf ("\nВведите последовательно строки массива строк [%d] размером не более 20 символов: \n" , MasSize);

for (int i =0 ; i < MasSize ; i++ )
{ char Buf[80] = "";
  // scanf ("%s", pStrMas + i * 20);
  scanf ("%s", Buf);
  // Контроль длины введенной строки
  if (strlen(Buf) > 20)
  {
    printf ("Введенная строка превышает размер 20 (%d): \n", strlen(Buf) );
    i--;
  }
  // scanf ("%s", &pStrArray[ i * 20]);
else
{
  strcpy(&pStrArray[ i * 20], Buf );
};
}
printf ("\n" );
// Распечатка массива
printf ("Порядок прямой\n" );
// Распечатка массива строк
printf (".....\n" ); // Рамка

for (int i =0 ; i < MasSize ; i++ )
//printf ("Строки массива % d - %s\n", i , pStrMas + i * 20);
{
  printf ( ". [%d ] . %10s .\n", i , &pStrArray[ i * 20] );
}

```

```

}
// Распечатка массива строк
////////////////////////////////////
printf ( ".....\n" ); // Рамка
printf ( "Порядок Обратный\n" );
printf ( ".....\n" ); // Рамка

for (int i = MasSize - 1 ; i >= 0 ; i-- )
{
    printf ( ".  [ %d ] .  %10s  \n", i , &pStrArray[ i * 20] );
};
printf ( ".....\n" ); // Рамка
printf ( "\n" );
////////////////////////////////////
Получим результат:
Введите размер массива строк:
3

```

Введите последовательно строки массива строк [3] размером не более 20 символов:

```

111
222
333

```

Порядок прямой

```

.....
.  [ 0 ] .          111 .
.  [ 1 ] .          222 .
.  [ 2 ] .          333 .
.....

```

Порядок Обратный

```

.....
    [ 2 ] .          333
    [ 1 ] .          222
    [ 0 ] .          111
.....

```

5.5. 5.5 Изменение порядка символов в строке

Изменить порядок расположения символов в строке (**Str5**[10]), заданной при описании ее в программе с помощью инициализации. Порядок замен определяется вариантом. Исходную строку и результат изменения распечатать. Строка заканчивается нулевым символом (/0), расположение которого не меняется при изменении порядка символов. Алгоритм замен нужно уметь продемонстрировать преподавателю в отладчике пошаговом режиме. Распечатать исходную строку и результат (строку) после изменения порядка.

```

// 5.5 Изменение порядка символов в строке на обратный
char StrPer[] = "Линейка";
char cTemp;
RazmS = strlen(StrPer); // Размер строки - вычисление
printf ( "Исходная строка =\n%s !\n", StrPer );
for (int i = 0 ; i < RazmS/2 ; i++)
{
    // Обмен символов (Swap) Крайние символы меняются местами
    cTemp = StrPer [ i ];
    StrPer[ i ] = StrPer[ RazmS - i - 1 ];
}

```

ОП ГУИМЦ 2024 ЛРН№4

```
StrPer[ RazmS - i -1] = cTemp ; };
StrPer[RazmS] = '\0';
printf ("Результирующая строка =\n%s !\n", StrPer);
```

Получим результат:

Исходная строка =

Линейка !

Результирующая строка =

акйенил !

Для продолжения нажмите любую клавишу . . .

5.6. 5.6 Динамические строки

Описать указатель на строку (**pStrD**). Выделить для этой строки динамическую память в 40 символов. Скопировать в эту строку большую строку из п.п. 5.1 (**FIO**). Распечатать исходную строку как массив (в цикле посимвольно!) и как динамическую строку через указатель. Изменить в данной строке первую букву отчества и снова распечатать. Освободить оперативную память, выделенную для динамической строки.

```
#include <malloc.h>
```

```
...
```

```
// 5.6 Динамические строки
```

```
{ char FIO[56];
```

```
// Копирование и слияние без контроля
```

```
strcpy(FIO, "Петров");
```

```
strcat(FIO, " "); // Нужно для пробела между фамилией и именем
```

```
strcat(FIO, "Петр");
```

```
strcat(FIO, " "); // Нужно для пробела между и именем и отчеством
```

```
strcat(FIO, "Иванович");
```

```
char * pStrD; // 5.6 Указатель на динамическую строку
```

```
pStrD = (char *) calloc (40 , sizeof(char) ); // размер 40 символов
```

```
//strcpy(StrPer, "Пример!!!");
```

```
strcpy (pStrD, FIO);
```

```
printf ("Динамическая строка =%s !\n", pStrD);
```

```
pStrD[12] = 'Ж';
```

```
free(pStrD); }; // Освобождение ДП под строку
```

Получим результат:

Динамическая строка =Петров Петр Иванович !

Динамическая строка (измененная) =Петров Петр Жванович !

6. 6. Варианты заданий для студентов СУЦ.

Варианты заданий приведены ниже. Номер варианта должен соответствовать номеру студента в групповом журнале.

п/п (Вар №)	Символы , для замены (зад. 5.3)	Символ, на который идет замена (зад.5.3).	Порядок вывода массива строк	Порядок замен символов в строке (Str5)	Поиск экстремум а в строке (д.т.)	Критерий величины строки при сортировке (д.т.)
1.	абвгде-+	*	Прямой	На обратный (первый символ становится последним и т.д.)	Минимум	strlen

2.	иклмн"№	пробел	Обратный (сначала выводиться последняя строка)	Четные меняют нечетные	Максимум	strcmp
3.	опрст(&)	\$	Прямой	На обратный	Минимум	strlen
4.	хццш]!}	*	Обратный	Четные - нечет	Максимум	strcmp
5.	абвгде-+	пробел	Прямой	На обратный	Минимум	strlen
6.	иклмн"№	\$	Обратный	Четные - нечет	Максимум	strcmp
7.	опрст(&)	*	Прямой	На обратный	Минимум	strlen
8.	хццш]!}	пробел	Обратный	Четные - нечет	Максимум	strcmp
9.	абвгде-+	\$	Прямой	На обратный	Минимум	Strlen
10.	иклмн"№	\$	Обратный	Четные - нечет	Максимум	strcmp
11.	опрст(&)	#	Прямой	На обратный	Минимум	strlen

7. 7. Дополнительные требования для студентов СУЦ (д.т.).

Для продвинутых студентов, по желанию, можно построить программу с дополнительными требованиями. Дополнительные требования выполняются в дополнение основным требованиям ЛР.

7.1. 7.1 Создание своей функции SubString(подстрока)

Спроектировать, реализовать и отладить функцию выбора подстроки из строки. В качестве параметров указываются: исходная строка, начальный символ, число символов. Предусмотреть вариант функции с 3-мя и 4-мя параметрами (Последний задает указатель подстроки). предусмотреть полный контроль входных параметров. Продемонстрировать ее использование в различных режимах. Функцию описать в отдельном заголовочном файле и использовать прототип в основной программе.

7.2. 7.2 Создание функции SwapString для строк равных по длине

Спроектировать, реализовать и отладить функцию взаимной замены (Swap) строк. В данной функции предположить, что строки имеют одинаковый размер символьных массивов, в котором они записаны. Строки не являются динамическими. Продемонстрировать ее использование в различных режимах. Функцию описать в отдельном заголовочном файле и использовать прототип в основной программе.

7.3. 7.3 Изменение порядка символов в строке

Изменить порядок расположения символов в строке (**Str5[10]**), заданной при описании ее в программе с помощью инициализации. Порядок замен определяется вариантом. Исходную строку и результат распечатать. Строка заканчивается нулевым символом, расположение которого не меняется. Алгоритм замен нужно уметь продемонстрировать преподавателю в отладчике в пошаговом режиме.

7.4. 7.4 Сортировка массива строк

Выполнить в двойном цикле сортировку массива строк по возрастанию. Массив задается в программе инициализацией при описании. Сортировка должна быть выполнена в алфавитном порядке по возрастанию. Для оценки возрастания использовать функцию из таблицы вариантов (**strlen** или **strcmp**). Для сортировки (обмена строк между собой) нужно использовать свою функцию **SwapString**, разработанную выше. Массив строк вывести на экран до сортировки и после.

7.5. 7.5 Минимум и максимум в массиве строк

В массиве (**MasStr**), который был введен в п.п 5.4 данной ЛР найти минимальную строку и ее номер. Результаты распечатать. Критерий “величины” (веса - упорядоченности) строки задается вариантом и может быть одним из двух: либо число символов в строке (для подсчета использовать функцию **strlen**) или алфавитный порядок (для оценки “величины” использовать функцию **strcmp**). Значения минимума и его номер распечатать.

7.6. 7.6 Замена подстроки в строке

Спроектировать, реализовать и отладить функцию замены одной подстроки другой подстрокой в большой строке. Например, по всей строке одна фамилия заменяется на другую фамилию. Использовать библиотечные функции для работы со строками. Продемонстрировать ее использование в различных режимах. Функцию описать в отдельном модуле *.cpp и использовать прототип в основной программе.

7.7. 7.7 Функция сравнение строк

Спроектировать, реализовать и отладить функцию сравнения строк, аналогичную **strcmp**. Продемонстрировать ее использование в различных режимах.

7.8. 7.8 Создание функции **DSwapString** для динамических строк.

Спроектировать, реализовать и отладить функцию взаимной замены (**Swap**) строк. Строки могут быть разной длины. Строки являются динамическими, то есть под них выделена динамическая память. Использовать функции **malloc** и **free** для работы с динамической памятью. Продемонстрировать использование функции в различных режимах. Функцию описать в отдельном заголовочном файле и использовать прототип в основной программе.

7.9. 7.9 Изучение библиотечных функций.

Изучить библиотечные функции: **strftime**, **strchr** и **_fcvt**. Познакомиться с документацией, привести примеры использования данных функций. Познакомиться с библиотеками: **string.h**, **malloc.h** и **stdlib.h**. Для этого помимо документации можно прочитать эти заголовочные файлы.

Примечание: Для того, чтобы ЛР была зачтена с выполнением дополнительных требований достаточно выполнить любые три задания этого раздела(см. выше).

8. 8. Контролируемые требования ЛР.

- Создание большой строки п.п 5.1
- Строка с инициалами п.п 5.2
- Замена символов и подсчет в строке п.п 5.3
- Ввод и вывод массива строк п.п 5.4
- Изменение порядка символов в строке п.п 5.5
- Динамические строки п.п 5.6
- **Блок-схемы для п.п.5.4,5.5**

9. 9. Демонстрация, защита ЛР и отчет по ЛР.

После выполнения всех необходимых шагов по ЛР, работающую программу нужно продемонстрировать преподавателю, проводящему ЛР, о чем он в журнале делает отметку. Далее студент на основе шаблона и примера оформляет отчет по ЛР. После оформления отчета, который может быть представлен преподавателю в электронном виде, выполняется защита ЛР. Студент дает ответы на вопросы по отчету и на контрольные вопросы приведенные ниже. ЛР считается полностью зачтенной, если выполнены все перечисленные требования и действия: демонстрация, отчет и защита ЛР.

10. 10. Контрольные вопросы по ЛР.

1. Как описываются строки в языке СИ?
2. Для чего нужны строки в языках программирования?
3. Как инициализировать строку?
4. Какой символ записывается в конце строки?
5. Что такое **Null Terminated String**?
6. Как скопировать строку в СИ?
7. Как определить длину строки?
8. Как определить длину символьного массива, в котором записана статическая строка?
9. Как слить две строки?
10. Как ввести строку с клавиатуры?
11. Как вывести строку на экран?
12. Какие уровни ввода/вывода вы знаете?
13. Как выполняется форматированный ввод/вывод?
14. Как преобразовать число в строку и наоборот?
15. Как преобразовать символы строки к нижнему/верхнему регистру?
16. Как сравнить две строки?
17. Что такое динамическая строка, как ее получить?
18. Какие функции используются для выделения и освобождения памяти в СИ?
19. Какие библиотеки есть для работы со строками?
20. Какие коды могут использоваться для строк?
21. Какие элементы блок схем вы знаете?

11. 11. Литература.**Основная литература**

1. Список литературы, доступные книги и необходимые пособия для ЛР ОП размещены на сайте www.sergebolshakov.ru на страничке “2-й к СУЦ”. Пароль для доступа можно взять у преподавателя или старосты группы.

2. Керниган Б., Ритчи Д. К36 Язык программирования Си.\Пер. с англ., 3-е изд., испр. - СПб.: "Невский Диалект", 2001. - 352 с.: ил.
3. Касюк, С.Т. Курс программирования на языке Си: конспект лекций/С.Т. Касюк. — Челябинск: Издательский центр ЮУрГУ, 2010. — 175 с.
4. MSDN Library for Visual Studio 2005 (Microsoft Document Explorer – входит в состав дистрибутива VS. Нужно обязательно развернуть при установке VS VS или настроить доступ через Интернет.)
5. С.О.Бочков, Д.М.Субботин Язык программирования Си для персонального компьютера, М.: "Радио и связь", 1990.- 384 с.
6. Фридланд А.Я. Информатика и компьютерные технологии: Основные термины: Толк.слов.: Более 1000 базовых понятий и терминов. – 3-е изд., испр. и доп./ А.Я Фридланд, Л.С. Ханамирова, И.А. Фридланд – М.:ООО «Издательство Астрель»: ООО «Издательство АСТ», 2003. - 272 с.

Дополнительная литература

7. Общее методическое пособие по курсу для выполнения ЛР и ДЗ (см. на сайте 1-й курс www.sergebolshakov.ru) – см. кнопку в конце каждого раздела сайта!!!
8. Другие методические материалы по дисциплине с сайта www.sergebolshakov.ru.
9. Конспекты лекций по дисциплине “Основы программирования”.
10. Подбельский В.В. Язык Си++: Учебное пособие. – М.: Финансы и статистика, 2003.
11. 5. Подбельский В.В. Стандартный Си++: Учебное пособие. – М.: Финансы и статистика, 2008.
12. Г. Шилдт “С++ Базовый курс”: Пер. с англ.- М., Издательский дом “Вильямс”, 2011 г. – 672с
13. Г. Шилдт “С++ Руководство для начинающих” : Пер. с англ. - М., Издательский дом “Вильямс”, 2005 г. – 672с
14. Г. Шилдт “Полный справочник по С++”: Пер. с англ.- М., Издательский дом “Вильямс”, 2006 г. – 800с
15. Бьерн Страуструп "Язык программирования С++"- М., Бином, 2010 г.