

**Методические указания к лабораторной работе № 3 по курсу  
ОСНОВЫ ПРОГРАММИРОВАНИЯ  
ГУИМЦ**

**"Массивы и указатели"  
( 4 часа)**

**СОДЕРЖАНИЕ**

СОДЕРЖАНИЕ .....	2
1. 1. Цель лабораторной работы №3 по курсу ОП(Основы программирования) .....	4
2. 2. Порядок выполнения лабораторной работы .....	4
3. 3. Основные понятия.....	4
3.1. 3.1 Массивы .....	4
3.2. 3.2 Инициализация массивов.....	5
3.3. 3.3 Указатели. Понятие. ....	6
3.4. 3.4 Указатели. Описание.....	6
3.5. 3.5 Указатели на массивы. ....	7
3.6. 3.6 Ввод вывод через указатель.....	7
3.7. 3.7 Адресная арифметика. ....	8
3.8. 3.8 Указатели на указатели. ....	9
3.9. 3.9 Указатели на строки.....	9
3.10. 3.10 Массивы указателей.....	10
3.11. 3.11 Ввод массива.....	10
3.12. 3.12 Печать массива .....	11
3.13. 3.13 Многомерные массивы .....	11
3.14. 3.14 Динамическая память.....	12
3.15. 3.15 Отладка программ. ....	14
3.16. 3.16 Модули .....	15
3.17. 3.17 Библиотеки функций.....	16
3.18. 3.18 Библиотеки функций и шаблонов классов: RTL, STL, MFC, ATL .....	16
3.19. 3.19 Болванка для главного модуля с математической библиотекой .....	17
4. Примеры по теме ЛР № 3 .....	18
3.20. 4.1 Примеры описаний и инициализации массивов .....	18
3.21. 4.2 Сумма элементов массива .....	18
3.22. 4.3 Поиск максимума/минимума в массиве и его номера.....	19
3.23. 4.4 Ввод и распечатка массива .....	20
3.24. 4.5 Инициализация массива в программе и определение отдельных сумм положительных и отрицательных элементов .....	20
3.25. 4.6 Сортировка массива по убыванию.....	21
3.26. 4.7 Динамические массивы и случайные числа .....	21
3.27. 4.8 Указатели и динамические массивы .....	22
3.28. 4.9 Итерационный цикл расчета значения функции на основе ряда .....	22
4. 5. Контрольные задания для ЛР №3. ....	24
4.1. 5.1 Создание консольного проекта .....	24
4.2. 5.2 Описание массивов.....	24
4.3. 5.3 Ввод и вывод массива. Вычисление суммы элементов .....	24
4.4. 5.4 Печать массива столбиком .....	25
4.5. 5.5 Определение минимума в массиве и его номера .....	25

## ОП ГУИМЦ 2024 ЛР№3

4.6. 5.6 Инициализация массива в программе и определение отдельных сумм положительных и отрицательных элементов .....	25
4.7. 5.7 Использование указателя для массива .....	26
4.8. 5.8 Вычисление и запоминание значений массива из значений вычисленного ряда.	26
4.9. 5.9 Заполнение и сумма в динамическом массиве через указатель. ....	27
4.10. 5.10 Двумерные массивы. ....	27
Блок-схемы заданий .....	27
5. 6. Контролируемые требования. ....	27
6. 7. Варианты заданий для студентов ГУИМЦ. ....	27
7. 8. Дополнительные требования для студентов СУЦ (д.т.). ....	29
7.1. 8.1 Ввод и вывод двумерного массива. Вычисление суммы его элементов. ....	30
7.2. 8.2 Определение максимума в двумерном массиве и координат максимума .....	30
7.3. 8.3 Инициализация двумерного массива .....	30
7.4. 8.4 Вычисления значений полинома .....	30
7.5. 8.5 Описание и инициализация двумерных массивов и умножение матриц .....	30
7.6. 8.6 Доказательство расположения массива. ....	30
7.7. 8.7 Блок-схемы заданий .....	30
8. 9. Демонстрация, защита ЛР и отчет по ЛР. ....	31
9. 10. Контрольные вопросы по ЛР. ....	31
10. 11 Литература. ....	32

**1. 1. Цель лабораторной работы №3 по курсу ОП(Основы программирования)**

Целью данной ЛР по дисциплине ОП является получение начальных знаний и умений для использования массивов и указателей на языке программирования СИ. Студенты используют консольные проекты и отлаживают программы в среде программирования MS VS 2005/2008/2010. Студенты знакомятся со способами описания и использования масс и указателей в программах, 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 (раздел 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 с массивами и указателями на языке программирования СИ.

**3.1. 3.1 Массивы**

Переменная, которая может сохранять только одно значение, называется простой переменной. Для разработки сложных программ часто недостаточно одних простых переменных, так как в противном случае простых переменных было бы очень много с разными именами, и программист может легко запутаться в таких наименованиях. Поэтому в языках программирования введено понятие массивов (упорядоченных наборов) однородных (одного типа) переменных, доступ к которым выполняется с указанием номера отдельного элемента (индекса). Такую переменную называют также переменной с индексами или элементом массива.

Таким образом, массив – это совокупность однотипных переменных, расположенных в порядке возрастания номера (от 0 до величины, определяющей его размер). При описании массива помимо имени и типа для массивов необходимо указать его размер и размерность массива (возможное число индексов). В общем случае массив описывается так:

**<тип массива > <имя массива> [<размер массива>]... [<размер массива>] = <инициализация массива>;**

**<тип массива >** - стандартный или пользовательский тип СИ/СИ++

**<имя массива>** - допустимое и уникальное в блоке имя в языке СИ/СИ++.

**<инициализация массива>** - задание начальных значений элементов массива

Например:

```
#define MAX 20
```

```
int iMas [10]; // описан массив iMas целого типа, содержащий 10 элементов ( номера 0:9)
char sMas [MAX]; // массив sMas символьного типа, содержащий MAX элементов
```

Размерность массива задается целой константой, переменной константного типа (**const**) или переменной этапа компиляции, и при таком описании размерность массивов не может изменяться во время работы программы. Номера массива начинаются с нуля (0), поэтому последний элемент массива имеет номер (точнее индекс) на единицу меньший, чем размерность массива (в нашем случае 9). При обращении к элементам массива можно указать, как константу, так и переменную целого типа, так и выражение целого типа.

Например:

```
iMas [5] = 3; // элементу массива iMas с номером 5 (6-му) присваивается 3
sMas [i + 5] = 'A'; // элементу массива sMas с номером i (i+4) присваивается символ 'A'
```

Массивы могут иметь более одного измерения, при этом указываются значения размерности по каждому измерению отдельно в квадратных скобках:

**<тип массива> <имя массива> [<размерность массива1>][<размерность массива2>] ...;**

Например, двумерный массив может быть описан так:

```
int iDMas [5][5]; // двумерный массив iMas целого типа, содержащий 5*5 элементов
```

При использовании переменных массивов с размерностью более 1-й должен быть указан номер (индекс) по каждому измерению (переменная с индексами – определяет в каждый момент времени один элемент массива):

```
int i=1, j=5;
```

```
iDMas [i][j] = 10; // элементу iMas с номерами i и j по каждому измерению присваивается 10
```

### 3.2. 3.2 Инициализация массивов

При описании массива можно задать начальные значения его элементов. Это выполняется так для одномерных массивов с заданным размером:

**<тип><имя массива>[<размер>]= {<выражение>, <выражение>, ...} }**

или без явного задания размера:

**<тип><имя массива>[] = {<выражение>, <выражение>, ...} }**

Во втором случае размер массива определяется автоматически по числу заданных выражений. В выражениях могут быть использованы: целочисленные переменные, значения которых задано к моменту описания данного массива. Если заданный при описании размер превышает число выражений, то последние элементы инициализируются нулем. Если число выражений меньше чем заданный размер массива, то выдается ошибка.

Примеры:

```
int iMas1[4] = {3,3,2,4}; // Инициализация массива – 4 элемента
int iMas1[] = {3,3,2,4,5}; // Инициализация массива – 5 элементов
int iMas1[6] = {3,3,2,4}; // Ошибка!
```

```
int elem = 3;
int iMas1[4] = {elem, elem}; // Массив <3,3,0,0,> – 4 элемента
```

В выражениях инициализации переменные должны быть заранее вычислены или инициализированы.

Массивы нужны для того, чтобы сделать программу более короткой, для обработки больших объемов данных и для обеспечения динамической настройки программ. Число измерений массивов в С и С++ формально не ограничивается.

Размерность массива в каждом измерении может быть задана (как сказано выше): целочисленной константой, переменной этапа компиляции (`#define`), константной переменной целого типа и целой переменной (для динамических массивов). При инициализации массивов фиксированным значением констант инициализации, его размер может быть не указан, он определяется автоматически. Рассмотрим еще примеры для разных способов задания размерности массива:

```
#define N 10 // Переменная этапа компиляции
...
int Mas[N]; // Описание массива размерностью 10
const int NMax = 5; // константная переменная
int MasS[NMax]; // Описание массива размерностью 5
```

Инициализация массивов заключается в задании начальных значений элементов массива:

```
int MasIni[] = {1,2,3,4}; // Описание и инициализация массива // и
// автоматический размер = 4
```

Инициализация двумерных массивов:

```
int C[3][6] = { { 1,2,3,4,5,6}, { 1,2,3,4,5,6}, { 1,2,3,4,5,6} };
```

В программе можно динамически определить число элементов в массиве:

```
int iMas1[] = {3,3,2,4,5,};
int Razm = sizeof(iMas1)/sizeof(int); // Вычисление размера массива
```

### 3.3. 3.3 Указатели. Понятие.

Указатель – это переменная специального типа, которая содержит адрес памяти, где расположена другая переменная (см. раздел ниже) или массив переменных (Этот фрагмент раздела можно пропустить, познакомится подробнее с указателями и вернуться к нему несколько позже). По - умолчанию, само имя массива трактуется как указатель на его начало. Проиллюстрируем это примерами:

Примеры работы с указателями:

```
int *pIntVar; // Указатель на целую переменную
pIntVar = &i; // Вычисление указателя
int iVar = *pIntVar; // чтение по адресу iVar = 5
*pIntVar = 10; // запись по адресу i = 10
int * PtrMas; // Указатель на целую переменную
```

Примеры работы с указателями на массивы и их элементы:

```
int MasP[]={ 1,2,3}; // Описание массива
PtrMas = MasP; // Указателю присваивается указатель на начало массива
int m = PtrMas[1]; // m = 2!
PtrMas = &MasP[0]; // новое вычисление указателя: !!! MasP == &MasP[0]
int k = PtrMas[2]; // k = 3!
int i=5; // целая переменная
```

Имя массива (`MasP`) – автоматически является указателем на его первый элемент (`&MasP[0]`)

### 3.4. 3.4 Указатели. Описание.

При описании указателей перед именем переменной ставится звездочка:

<тип> \* <имя> [=<адресное выражение>]

Примеры описания указателя:

```
int * PtrMas; // Указатель на целую переменную
int **ppInt; // Указатель на указатель на переменную типа int
```

Для работы с указателями используются два типа операций:

- Операция именования - вычисления адреса (&) и
- Операция разыменования – получения значения по адресу (\*).

```
int * PtrMas; // Указатель на целую переменную
int **ppInt; // Указатель на указатель на переменную типа int
int i = 5;
int pi = &i; // вычисления адреса переменной i
int b = *pi; // взятие значение переменной i по адресу pi (b = 5)
```

### 3.5. 3.5 Указатели на массивы.

Для работы с массивом может задаваться указатель на массив, который вычисляется как адрес на первого элемента массива. Тогда имя указателя может использоваться в индексном выражении.

Примеры:

```
int Mas[] = {1,2,3,4,5}; // Описание массива
int * pMas1; // Указатель на int
pMas1 = &Mas[0]; // Вычисление указателя как адреса первого элемента массива
pMas1 = Mas; // Имя массива == указателю на первый элемент (равно &Mas[0])
i = pMas1[2]; // Индексное выражение с указателем, результат i = 3
```

При вводе функцией **scanf**, для указателя задается выражение на основе адресной арифметики:

```
scanf("%d", &Mas[2]); // Ввод третьего элемента массива Mas[2]
scanf("%d", pMas1 + 3); // Ввод четвертого элемента массива Mas[3]
scanf("%d", pMas1); // Ввод первого элемента массива Mas[0]
scanf("%d", (int *)pMas1[3]); //Попытка ввода - Ошибка этапа выполнения pMas1[3]– это не адрес!!!
```

Для ввода через указатель и индекс i в цикле (Д.П. ) нужно использовать адресное выражение <указатель + индекс>:

```
i = 2;
scanf("%d", pMas1 + i); // Ввод третьего элемента массива Mas[2]
```

### 3.6. 3.6 Ввод вывод через указатель.

Вывод через указатель:

```
printf ("\nПечать по указателю = %d\n", *pInt );
printf ("\nПечать значения указателя (адреса) = %p\n", pInt );
double *pDVar; // Указатель на переменную типа double
double dVar = 5.5;
pDVar = &dVar ; // Вычисление указателя
printf ("\nПечать по указателю (double) = %7.2lf\n", *pDVar );
```

Ввод через указатель:

```
double *pDVar; // Указатель на переменную типа double
double dVar = 5.5;
pDVar = &dVar ; // Вычисление указателя
printf ("\nПечать по указателю (double) = %7.2lf\n", *pDVar );
```

Результат работы фрагмента:

Печать по указателю = 25

```

Печать значения указателя (адреса) = 001DF7A8
Печать значения указателя (адреса) = 001DF7A8
Печать по указателю (double) = 5.50
Введите целое:77
Печать введенного по указателю = 77

```

### 3.7. 3.7 Адресная арифметика.

Если в целочисленном выражении используется указатель, то вычисление выполняется по правилам адресной арифметики:

1. Результатом вычисления выражения будет адрес.
2. В сложении каждая единица при сложении или вычитании соответствует размеру типа объявленного для указателя (int = 4, проверьте sizeof). Примеры:

```

// Адресная арифметика
int * pOld= pInt;
printf ("\nАдресная арифметика:\n" );
printf ("\nПечать значения указателя (до) = %p\n", pInt );
pInt++; // увеличение на 4 - размер int
printf ("\nПечать значения указателя (после) = %p \nИзменение в адресах: %d в числах: %d\n", pInt , pOld-pInt , (pOld-pInt)* sizeof(int) );
i = 2;
pInt +=i; // увеличение на 8 - размер int * 2
pInt = pInt - 1 ; // уменьшение на 4
pInt +=i*2; // увеличение на 16 i = 8 * 2

```

Результат:

**Адресная арифметика:**

Печать значения указателя (до) = 0028FC40

Печать значения указателя (после) = 0028FC44

Изменение в адресах: 1 в числах: 4

Забегая немного вперед, проиллюстрируем использование указателей для работы с динамическими массивами (см. динамическая память). Проиллюстрируем это на примере. В указателе **PtrMas** запоминается адрес области динамической памяти, выделенной оператором **new** (операция C++). Далее динамический массив заполняется в цикле. Затем с использованием операции разыменования (\*) мы получим новое значение из массива (в переменную l). Оно равно 3.

```
PtrMas = new int [5]; // Динамическое выделение памяти
```

```
for ( int k = 0 ; k < 5 ; k++)
```

```
PtrMas[k] = k + 1; // Динамическое заполнение массива числами от 1 до 5
```

```
int l = 2 ;
```

```
*(PtrMas + l) = l + 1; //Использование указателя для занесение про [2] значения 3 (2+1)
```

```
l = PtrMas[l] ; // l = 3
```

Или

```
for ( int k = 0 ; k < 5 ; k++)
```

```
PtrMas[k] = k + 1; // Динамическое заполнение массива числами от 1 до 5
```

```
int l = 2 ;
```

```
*(PtrMas + l) = l + 1; //Использование указателя для занесение про [2] значения 3 (2+1)
```

```
l = PtrMas[l] ; // l = 2 - 3-й элемент массива = 3
```

```
delete []PtrMas;
```

```
// выделение памяти с помощью библиотеки <malloc.h >
```

```
PtrMas = (int *)malloc(sizeof (int)*5);
```

```
for ( int k = 0 ; k < 5 ; k++)
```

```
PtrMas[k] = k + 10; // Динамическое заполнение массива числами от 1 до 5
```

```
l = PtrMas[l] ; // l = 13
```

```
free(PtrMas);
```



### 3.8. 3.8 Указатели на указатели.

Кроме использования массивов, есть и другой способ, чтобы сделать программу динамически настраиваемой во время выполнения. Это переменные специального типа - указатели. Такие переменные содержат в качестве значения не число, а адрес другой переменной (в оперативной памяти). При описании таких переменных нужно указать специальный знак – звездочка (“\*”). Кроме того, должно быть указан тип переменных, на которые данный указатель может ссылаться. Можно объявить и массив указателей и выполнить предварительную инициализацию указателя. Описание указателя:

**<тип указателя> \* <имя указателя> [ = <значение для инициализации>];**

Например:

```
// Указатели
int /*i , */j , k ;
int *pInt; // Указатель на переменную типа int
int **ppInt; // Указатель на указатель на переменную типа int
int *pMas[10]; // Массив указателей
int aI =5; // Простая переменная
int *pInit = &aI; // инициализация указателя
```

Для работы с указателями используются две специальные операции: операция именования (“&”) и операция разыменования (“\*”). Операция именования используется для вычисления значения указателя – адреса переменной или выражения. Операция разыменования используется для получения значения переменной, адрес которой задан данным указателем. Примеры операций:

```
/// Задание значений и адресов
j = 15;
pInt = &j; // именование - в указатель записывается адрес
i = *pInt; // разыменование – берем значение переменной по указателю (j)
ppInt = &pInt;
k = **ppInt; // двойное разыменование указателя
```

Еще примеры (обратите внимание на комментарии):

```
int * PtrMas; // Указатель на целую переменную
int **ppInt; // Указатель на указатель на переменную типа int
int i = 5;
int pi =&i; // вычисления адреса переменной i
int b = *pi; // взятие значение переменной i по адресу pi (b =5)
```

Операции с указателями могут использоваться многократно(\*\*):

```
int b = *pi; // взятие значение переменной i по адресу pi (b =5)
ppInt = &pi; // формирование указателя на указатель
int c = **ppInt; // Двойное разыменование
```

### 3.9. 3.9 Указатели на строки

Строки (СМ) и указатели на них. В Си нет отдельного типа строка (СМ). Для хранения текстовых переменных используются символьные массивы (char). Пример:

```
// Указатели на строки
char * pStr; // Указатель на строку
char Str[]="Это строка!";
pStr = Str; // Вычисление указателя
```

## ОП ГУИМЦ 2024 ЛР№3

```
printf ("\nПечать строки через указатель = %s\n", pStr );
pStr = &Str[0];    // Или так
printf ("\nПечать строки через указатель = %s\n", pStr );
```

Результат:

```
Печать строки через указатель = Это строка!
Печать строки через указатель = Это строка!
```

Имя этого массива одновременно является указателем. В функциях библиотеки строк и пользовательских функциях. Функции (СМ) и строки (СМ). Работа со строками будет рассмотрена в ЛР №4.

### 3.10. 3.10 Массивы указателей

Массив указателей объявляется так:

```
<тип> * <имя> [= <размер - выражение>] = {<список инициализации>;};
```

или

```
<тип> * <имя> [ ] = {<список инициализации>}
```

Примеры описаний массивов указателей:

//Работа с массивами указателей:

```
int i = 1;
int j = 1 , k = 2 , l = 3 , m = 4 ;
int * ipMas2[10]; // массив указателей на int без инициализации
int * ipMas[]={&j,&k,&l,&m}; // массив указателей на int с инициализацией
//Работа с массивами указателей:
ipMas[2] = &i; // Адресация ipMas[2] содержит адрес i ( i = 4 )
i = *(ipMas[3]); // Разыменование i = l ( i = 4 )
pInt = ipMas[2] ; //
*pInt = 5; // i = 5
```

Для упрощения работы с указателями, а также для удобной перегрузки операций в классах в C++ введено понятие ссылки. Ссылка также задает адреса других переменных и объектов, но явного использования операций именования и разыменования для ссылок не требуется. Более детально со ссылками вы познакомитесь в курсе Объектно - ориентированного программирования (см. литературу[6] и MSDN).

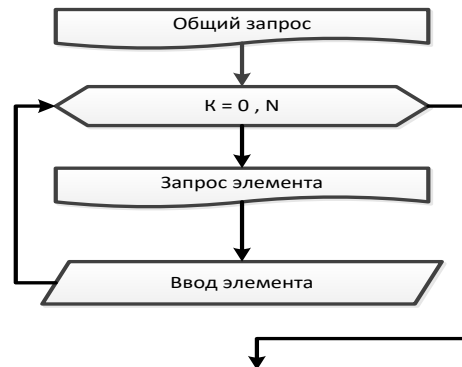
### 3.11. 3.11 Ввод массива

Для поэлементного ввода массива используется функция scanf. При этом для каждого элемента задается адрес (&MasInp[k]);

```
int MasInp[3]; // Массив для
// ввода
printf("Введите целый массив размером %d \n", sizeof(MasInp)/sizeof(int));
for (int k = 0 ; k < (sizeof(MasInp)/sizeof(int)) ; k++ ) {
printf("\nВведите элемент MasInp[%d]:", k);
scanf("%d" , &MasInp[k]);
} ;
```

Блок-схема фрагмента ввода массива:

Блок-схема



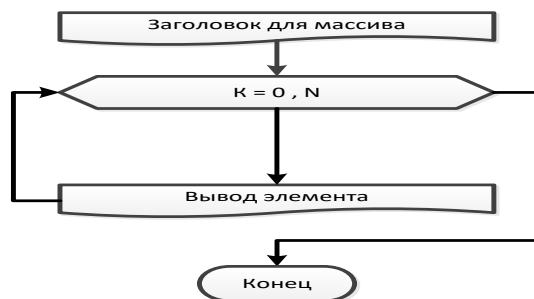
### 3.12. 3.12 Печать массива

Печать массива может быть выполнена в простом цикле с оператором `for`. Одновременно будем использовать программное определение размера массива (см. выше). Пример:

```
printf("Массив размером %d: \n", sizeof(MasInp)/sizeof(int));
for (int k = 0 ; k < (sizeof(MasInp)/sizeof(int)) ; k++ ) {
printf("\nЭлемент MasInp[%d] = %5d", k, MasInp[k]);
};
printf("\n");;
```

Блок-схема фрагмента печати приведена ниже:

Блок-схема



### 3.13. 3.13 Многомерные массивы

Массив, в котором задано более одной размерности называется многомерным (двумерным, трехмерным и т.д.). При описании многомерных массивов указывается несколько размеров:

**<тип> <имя> [= <размер – выражение1> ]... [= <размер – выражение2> ]...**

Пример:

```
int MasD[10][5]; // двумерный массив 10*5
```

Каждый из индексов многомерного массива задается в пределах от 0 до значения размер (у нас в примере: 0-9 и 0-4). При работе с индексной переменной многомерного массива переменная с индексом задается так:

**<имя массива> [= <индекс 1> ][<индекс 2> ]...**

Примеры переменных с индексами многомерных массивов:

```
MasD[1][2] = 6; // берем элемент с индексами 1 и 2
MasD[i][i+1] = 6; // берем элемент с индексами i и i+1
```

Инициализация многомерных массивов проводится построчно.

<тип> \* <имя> [= <размер - выражение>] = {{<список инициализации1>},...{<список инициализации2>}, ...};

Примеры инициализации многомерных массивов:

```
int C[3][6] = { { 1,2,3,4,5,6}, { 1,2,3,4,5,6},{ 1,2,3,4,5,6} };
```

Примеры инициализация и печать многомерных массивов через указатель:

```
int D[3][6] = { { }, { }, { } }; // Массив пустой
int MMas [3][3] = {{11,12,13},{21,22,23},{31,32,33}};
int * pMas= (int *)MMas;
// printf(" = %d \n" ,*pMas );
for (int i =0 ; i < 3; i++)
    for (int j =0 ; j < 3; j++)
        { printf("%d - %d = %d \n",i,j, MMas[i][j] );
          printf("%d - %d => %d \n" ,i,j,*((int *) (pMas + i*3 + j )) ); // Правильно
        }
};
```

Печать двумерного массива, как одномерного:

```
int * pMas= (int *)MMas;
for (int i =0 ; i < 9; i++)
    printf("%d - %d => %d \n" ,i,*((int *) (pMas + i )) );
```

Результат работы фрагментов программ печати:

```
0 - 0 = 11
0 - 0 => 11
0 - 1 = 12
0 - 1 => 12
0 - 2 = 13
0 - 2 => 13
1 - 0 = 21
1 - 0 => 21
1 - 1 = 22
1 - 1 => 22
1 - 2 = 23
1 - 2 => 23
2 - 0 = 31
2 - 0 => 31
2 - 1 = 32
2 - 1 => 32
2 - 2 = 33
```

Печать двумерного массива, как одномерного (вывод):

```
0 - 1 => 11
1 - 1 => 12
2 - 1 => 13
3 - 1 => 21
4 - 1 => 22
5 - 1 => 23
6 - 1 => 31
7 - 1 => 32
8 - 1 => 33
```

### 3.14. 3.14 Динамическая память

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, то есть его размерность в этом случае изменить нельзя. Поэтому приходится заранее рассчитывать максимально возможную размерность массива, которая приемлема для всех случаев. Это приводит к перерасходу памяти для отдельных

случаев. В СИ/СИ++ предусмотрен механизм работы с динамически выделяемой памятью (ДП). Это специальная область ОП, которой управляет ОС. Поэтому, для экономии памяти и для построения более универсальных программ часто используют динамически выделяемые переменные. В языке СИ++ с помощью специальных операций (**new** и **delete**) можно выделять и освобождать память во время выполнения программы. Такие данные называются динамическими. В языке СИ оперативная память выделяется с помощью специальных функций (alloc, malloc и т.д.). Динамические переменные располагаются в оперативной памяти в специальной области. Для работы с такими данными используют указатели и ссылки.

Для работы с этой памятью в СИ/СИ++ существует библиотека: <malloc.h>, в которой предусмотрены функции:

- malloc ()– захват ДП байтами,
- calloc ()– захват ДП блоками
- free – () освобождение ДП
- realloc ()– переопределение ДП
- и другие.

Для работы с ДП необходим предварительно объявить указатель такого типа, массив или переменную которого мы собираемся использовать:

Пример выделения динамической памяти для указателей и ссылок для СИ++:

```
int * piDyn = new int; // Выделяется область для одной целой переменной
int *piDMas = new int[10]; // Выделяется область для массива целых 10 переменных
int *piDMas = new int[i]; // Выделяется область для массива целых i переменных
```

Работа в СИ с указателями на динамические переменные и массивы выглядит так:

```
*piDyn = 5;
piDMas[0] = 5;
```

По завершению блока, в котором выделены динамические переменные их надо удалить специальным оператором **delete**:

```
delete piDyn; // Освобождение по указателю
delete []piDMas;
```

Кроме этого для работы с динамической памятью могут быть использованы функции работы с динамической памятью (библиотека **malloc.h**): выделение (**calloc** , **malloc**) и освобождения (**free**), оперативной памяти Пример:

```
#include <malloc.h>
...
int *pMasInt;
...
pMasInt = (int *) malloc ( 10 ); // выделить 10 байт
...
free( pMasInt ); // Освободить динамическую память
...
pMasInt = (int *) calloc ( 10 , sizeof (int)); // выделить для массива 10 блоков по размеру int
pMasInt[3] = 10; // Работа с динамическим массивом
int iTest = pMasInt[3] ;
...
free( pMasInt ); // Освободить динамическую память
```

Другие возможности для работы с библиотекой вы найдете в литературе и документации по СИ. С указателями можно работать в цикле (заполнение массива):

```
// выделение памяти с помощью библиотеки <malloc.h >
PtrMas = (int *)malloc(sizeof (int)*5);
```

ОП ГУИМЦ 2024 ЛРН№3

```
for ( int k = 0 ; k < 5 ; k++)
PtrMas[k] = k + 10; // Динамическое заполнение массива числами от 10 до 15
int l = PtrMas[3] ; // l = 13
free(PtrMas); };
```

С указателями можно работать в цикле (распечатка динамического массива):

```
// Печать
for ( int k = 0 ; k < 5 ; k++)
printf("PtrMas[%d] = %d\n",k, PtrMas[k] );
// Или через указатель -адрес
printf("PtrMas[%d] = %d\n",k, *(PtrMas + k) );      } ;
free(PtrMas);
```

В результате получим:

```
PtrMas[0] = 10
PtrMas[1] = 11
PtrMas[2] = 12
PtrMas[3] = 13
PtrMas[4] = 14
```

### 3.15. 3.15 Отладка программ.

В этом разделе повторяем возможности отладчика, так как демонстрация программы преподавателю должна быть выполнена в режиме отладчика. При разработке программ важную роль играет отладчик, который встроен в систему программирования. В режиме отладки можно проверить работоспособность программы и выполнить поиск ошибок самого разного характера. Отладчик позволяет проследить ход (по шагам) выполнения программы и одновременно получить текущие значения всех переменных и объектов программы, что позволяет установить моменты времени (и операторы), в которые происходит ошибка и предпринять меры ее устранения. В целом, отладчик позволяет выполнить следующие действия:

- Перекомпилировать все и сделать новую сборку (**F7**);
- Запустить программу в режиме отладки без трассировки по шагам (**F5**);
- Выполнить программу по шагам (**F10**);
- Выполнить программу по шагам с обращениями к вложенным функциям(**F11**);
- Установить точку останова (**BreakPoint – F9**);
- Выполнить программу до первой точки останова (**F5**);
- Просмотреть любые данные в режиме отладки в специальном окне (**locals**);
- Просмотреть любые данные в режиме отладки при помощи мышки;
- Изменить любые данные в режиме отладки в специальном окне (**locals** и **Watch**);
- Установить просмотр переменных в специальном окне (**Watch**);
- Просмотреть последовательность и вложенность вызова функций.

При выполнении всех лабораторных курса студенты должны активно использовать отладчик VS, знать его возможности и отвечать на контрольные вопросы, связанные с отладкой и тестированием программ.

**Примечание.** Подробное описание материала и понятий вы можете найти в литературе [1 - 6] или справочной системе MS VS. Кроме того, не пропускайте лекции по курсу. Не рекомендую безоговорочно верить материалам из сети Интернет (например, в Википедии), так как там в некоторых статьях есть ошибки!

### 3.16. 3.16 Модули

При проектировании и разработке программ применяется метод модульного программирования. Суть его заключается в том, что сложная программа разбивается на отдельные части (модули – не надо путать с учебными модулями). Каждый модуль разрабатывается отдельно и возможно разными программистами. Такой способ называют также декомпозицией. Совокупность модулей составляет проект программы. Модули бывают разных типов:

- Исходные модули, содержащие текст на языке программирования.
- Объектные модули получаются в результате компиляции программы.
- Исполнимые модули предназначены для выполнения программы.
- Исходные модули могут быть разных типов.

Основные исходные модули – это модули программ (\*.c, \*.cpp) и модули заголовочных файлов (\*.h, \*.hpp). Они включены в разные разделы дерева проекта программы.

Объектные модули формируются компилятором языка программирования (у нас C++) в том случае, если не было ошибок в программе. Объектные модули являются промежуточным звеном в создании программ, но не могут выполняться непосредственно. Подключаемые в программу библиотеки содержат объектные модули, поэтому не требуется их повторной компиляции. Объектные модули имеют расширение \*.obj.

Исполнимые модули предназначены для непосредственного выполнения на компьютере или подключения в выполняемую программу. Основные исполнимые модули имеют расширение \*.exe (или \*.com), поэтому операционная система может контролировать их запуск. Модули динамических библиотек имеют расширение \*.dll. Существуют и другие разновидности исполнимых модулей, зависящих от используемых технологий. Исполнимые модули формируются специальной программой системы программирования редактором связей (или компоновщиком). Редактирование связей заключается в проверке межмодульных связей по функциям и по данным и объединении их единый исполнимый модуль. Последовательный процесс обработки программы для получения исходного модуля представлен в методическом пособии в разделе 3[5].

Для объединения модулей функционально и по данным в C++ используются прототипы функций и описание внешних переменных. Покажем это на простом примере:

```
// Модуль 1
int i = 0; // Глобальная переменная
int Summ(int a, int b){}; // Описание функции в модуле 1
...
// Модуль 2
extern int i; // Описание внешних данных
int Summ(int , int ); // Прототип внешней функции
main(){
...
i = 5; // Использование внешних данных
S = Summ(2,2); // Вызов внешней функции
...
}
```

В модуле 2 используется переменная i, описанная как глобальная переменная в модуле 1. В модуле 2 используется функция **Summ**, описанная как в модуле 1. Для правильного объединения связей (работы редактора связей) используются описания внешних данных (**extern**) и задание прототипа функции **Summ**.



**3.17. 3.17 Библиотеки функций**

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 (имен - **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>
#include <process.h>
#include <stdio.h>
```

Стандартных библиотек 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.

**3.18. 3.18 Библиотеки функций и шаблонов классов: RTL, STL, MFC, ATL**

В СИ++ введены специальные классы для работы с массивами. Для их использования необходимо освоить основы Объектно-ориентированного программирования, что вам предстоит в дальнейших дисциплинах. Для общих сведений приводим здесь материал о назначении библиотек MS VS.

В системы программирования включаются различные библиотеки функций и классов. В MS VS включено несколько групп таких библиотек, которые постоянно развиваются и добавлялись по мере развития самой системы программирования. К сожалению, для разных языков и систем программирования пока нет единого стандарта, поэтому разрабатывать программные системы, используя библиотеки разных разработчиков, затруднительно. Однако существуют технологии и платформы, позволяющие решить эту проблему.

В MS VS предусмотрены следующие группы библиотек:

- C RTL - C Run-Time Libraries - стандартные библиотеки этапа выполнения.
- STL - Standard C++ Library – стандартные библиотеки C++.
- MFC - Microsoft Foundation Class Library – библиотеки классов MS.
- ATL - Active Template Library – библиотеки шаблонов.
- .NET Framework Class Library – библиотеки платформы .NET.

В C RTL включено много библиотек для работы со стандартными типами данных, файлами, строками и т.д. Эти библиотеки возникли относительно давно, но поддерживаются в настоящее время и представляют собой набор базовых средств программирования на языках C и C++.

В STL включено много библиотек, ориентированных на C++, и обеспечивающих работу с множеством классов, объектов и шаблонов. Эти библиотеки доступны всем пользователям и включают: библиотеки потокового ввода, вывода (cin, cout), работу с контейнерами объектов (vector, string, list, map, stack, set и многие другие.), работу с функциями из библиотек этапа выполнения (CRT).

В MFC включен широчайший набор классов и функций, позволяющих обеспечить программирование практически любых задач. Это библиотеки классов и функций: построения



приложений, в том числе и в среде Windows, оконного интерфейса, управления файлами, работы с контейнерными классами, графического интерфейса, поддержки различных технологий, в том числе и современных (версии библиотек постоянно обновляются), программирования коммуникаций, обработки документов и многое другое. В частности, в других ЛР, мы познакомимся с контейнерами типа CArray, CList и др.

В ATL представлен набор библиотек, поддерживающих АХ и СОМ технологий. Число доступных классов и шаблонов в этих библиотеках соизмеримы с набором библиотек MFC, они имеют дополнительные классы и классы, которые можно одновременно использовать совместно с MFC ("Shared by MFC and ATL"). В частности, в других ЛР, мы познакомимся с контейнерами типа CAtlArray, CAtlList и др.

.NET библиотеки представляют собой набор (иногда и более точно говорят платформу) библиотек для разработки программ – приложений самого универсального вида. В эту платформу включаются такие библиотеки, которые исключают необходимость использования устаревших библиотек (CTR и STL). Применение этой платформы позволяет сделать приложения более универсальными, охватывает большее число современных технологий и обеспечивает интеграцию с другими системами программирования, в том числе и на разных языках. В частности в этой платформе определены шаблоны классов Array, List и многие другие. .NET библиотеки широко используются для создания сайтов (ASP). Платформа .NET будет изучаться в отдельных дисциплинах.

Описания библиотек наиболее полно и с примерами в представлены [4]. Кроме того, описания библиотек CTR и STL вы найдете в [13]. Вся новую информацию об библиотеках можно также найти на сайтах MS. Хорошее знание существующих библиотек, их пополнения в новых версиях, несомненно, поможет вам стать профессионалом в программировании. В других ЛР цикла вы познакомитесь с составляющими других библиотек.

### **3.19. 3.19 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. Примеры по теме ЛР № 3**

Вторая часть задания, помимо первой связанной с изучением теоретического раздела, заключается в том, чтобы испытать в проекте СИ уже отлаженные программы и фрагменты программ. Нужно изучить технику этих программ и алгоритмы их работы. Это Вам понадобится для выполнения контрольных заданий. Возможно, что, осваивая теоретическую часть работы, вы уже на компьютере проверили выполнение фрагментов текста и применения массивов и указателей, тогда вам будет проще продемонстрировать их работу преподавателю. В дополнение к примерам, расположенным выше нужно испытать и изучить примеры расположенные ниже в этом разделе. Эти действия нужно обязательно сделать в отладчике.

Для выполнения этой части ЛР этого нужно создать пустой проект в MS VS (Test\_LR3), как описано выше, скопировать через буфер обмена в него текст данных примеров, отладить его и выполнить.

**3.20. 4.1 Примеры описаний и инициализации массивов**

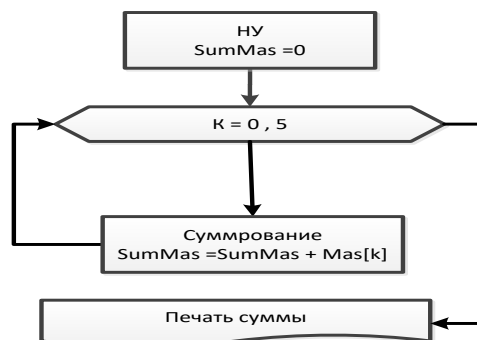
Здесь нужно проверить все примеры описания и инициализации массивов, приведенные в теоретической части ЛР. Проверьте операторы динамического определения размерности массива и выделения памяти под динамические переменные.

**3.21. 4.2 Сумма элементов массива**

Ниже приведена работающая программа, в которой вычисляется сумма элементов массива. Массив задан значениями инициализации при описании. Число элементов фиксировано.

```
// Описание массивов
int Mas[5] = { 1,2,3,4,5}; // элементы с индексами 0 ... 4
// Простой цикл с массивом
int SumMas = 0;
for (int k = 0 ; k < 5 ; k++ )
    SumMas = SumMas + Mas[k] ;
printf("\nСумма массива SumMas = %d \n" , SumMas);
```

Блок-схема



Попробуйте изменить значения в исходном массиве и выполнить вычисление заново в пошаговом режиме.

Результат вычисления:

**Сумма массива SumMas = 15**

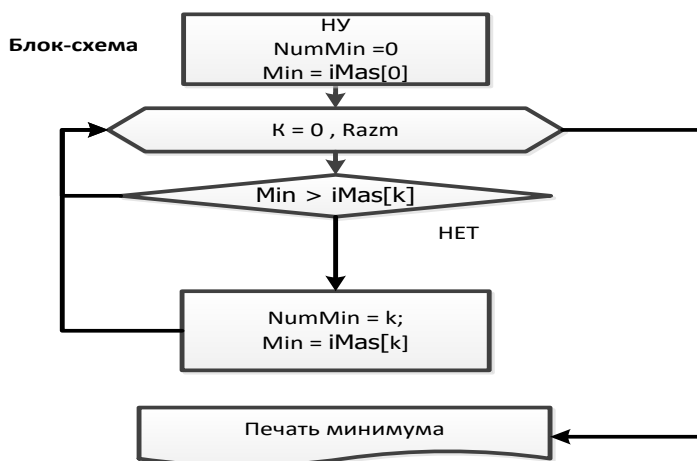
**3.22. 4.3 Поиск максимума/минимума в массиве и его номера**

Проверьте нижерасположенную программу поиска минимума и минимума в массиве.

```
// Дан массив iMas. Найти максимальный элемент и его порядковый номер.
int iMas[] = {1,22,3,-4,5,4,0};
int Max;
int Razm = sizeof(iMas)/sizeof(int);
// Распечатка массива
printf("\nЗадан массив iMas :\n");
for (int i = 0 ; i < Razm; i++)
    printf( "iMas[%2d] = %d\n", i , iMas[i]);
// Начальные условия цикла поиска максимума
int NumMax = 0;
Max = iMas[0];
// Поиск Max
for ( int i = 1; i < Razm; i++)
    if ( Max < iMas[i])
        { Max = iMas[i]; NumMax = i; };
// вывод результата
printf("\nВ массиве Max= %d его номер N = %d \n", Max ,NumMax);

// Начальные условия цикла поиска минимума
int iMas[] = {1,22,3,-4,5,4,0};
int NumMin = 0;
int Min = iMas[0];
for ( int i = 1; i < Razm; i++)
    if ( Min > iMas[i])
        { Min = iMas[i]; NumMin = i; };
// вывод результата
printf("\nВ массиве Min= %d его номер N = %d\n", Min ,NumMin);
В массиве Min= -4 его номер N = 3
```

Блок-схема фрагмента программы минимума:



**3.23. 4.4 Ввод и распечатка массива**

```
// Ввод и распечатка массива
int MasInp[3]; // Без инициализации
printf("Введите целый массив размером %d \n", sizeof(MasInp)/sizeof(int));
for (int k = 0 ; k < (sizeof(MasInp)/sizeof(int)) ; k++ ) {
    printf("\nВведите элемент MasInp[%d]:", k);
    scanf("%d" , &MasInp[k] );
};
// Распечатка введенного массива
printf("Массив размером %d: \n", sizeof(MasInp)/sizeof(int));
for (int k = 0 ; k < (sizeof(MasInp)/sizeof(int)) ; k++ ) {
    printf("\nЭлемент MasInp[%d] = %5d", k,MasInp[k]);
};
printf("\n");
```

**3.24. 4.5 Инициализация массива в программе и определение отдельных сумм положительных и отрицательных элементов**

Описать массив действительных(double) чисел **MasD** (6 элементов) и выполнить его инициализацию при описании. Константы инициализации массива должны быть разных знаков. Построить один цикл для вычисления отдельных сумм положительных и отрицательных элементов. Результаты вычислений сумм вывести на экран.

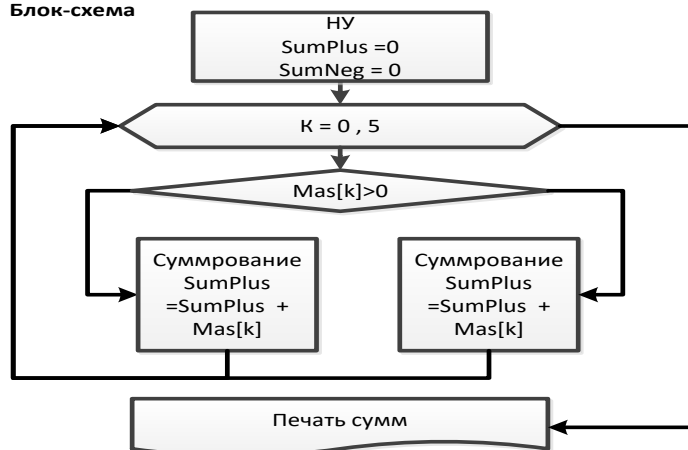
```
double MasD[] = {3.0 , -5.1, 1.0 , -7.2};
double SumPlus = 0.0;
double SumNeg = 0.0;
int Razm = sizeof(MasD)/sizeof(double);
for (int i = 0 ; i < Razm ; i++)
{
    if (MasD[i] > 0.0) SumPlus+=MasD[i];
    else SumNeg+=MasD[i];
// Вывод результата
};
printf("\nСумма положительных = %lf \n", SumPlus);
printf("Сумма отрицательных = %lf \n", SumNeg);
```

В результате получим:

Сумма положительных = 4.000000

Сумма отрицательных = -12.300000

Блок-схема



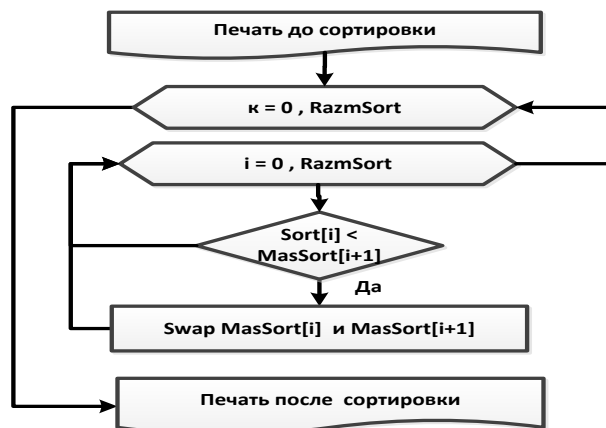
**3.25. 4.6 Сортировка массива по убыванию**

```

int MasSort[] = {3,5,11,0,2};
printf("Массив MasSort до сортировки размером %d: \n", sizeof(MasSort)/sizeof(int));
int RazmSort = sizeof(MasSort)/sizeof(int);
for (int k = 0 ; k < (sizeof(MasSort)/sizeof(int)) ; k++ ) {
    printf("\nЭлемент MasSort[%d] = %5d", k, MasSort[k]);
}
printf("\n");
// Сортировка
for (int k = 0; k < (RazmSort - 1) ; k++ ) {
    for (int i = 0; i < (RazmSort - 1) ; i++ ) {
        if (MasSort[i] <= MasSort[i+1]) // убывание
        { // Swap
            int Temp;
            Temp = MasSort[i];
            MasSort[i] = MasSort[i+1];
            MasSort[i+1] = Temp;
        }
    }
}
// Печать
printf("Массив MasSort после сортировки размером %d: \n", sizeof(MasSort)/sizeof(int));
for (int k = 0 ; k < (sizeof(MasSort)/sizeof(int)) ; k++ ) {
    printf("\nЭлемент MasSort[%d] = %5d", k, MasSort[k]);
}
printf("\n");

```

Блок-схема сортировки имеет вид:



Результат сортировки:

Массив MasSort до сортировки размером 5:

```

Элемент MasSort[0] = 3
Элемент MasSort[1] = 5
Элемент MasSort[2] = 11
Элемент MasSort[3] = 0
Элемент MasSort[4] = 2

```

Массив MasSort после сортировки размером 5:

```

Элемент MasSort[0] = 11
Элемент MasSort[1] = 5
Элемент MasSort[2] = 3
Элемент MasSort[3] = 2
Элемент MasSort[4] = 0

```

Для продолжения нажмите любую клавишу . . .

**3.26. 4.7 Динамические массивы и случайные числа**

Выделяется память под динамический массив 10 целых элементов (PtrMas). Заполняется

## ОП ГУИМЦ 2024 ЛР№3

```
#include <malloc.h>
#include <stdlib.h>
printf("Динамический массив и Случайные числа: \n" );
// выделение ДП с помощью библиотеки <malloc.h> - указатель PtrMas
int *PtrMas = (int *)malloc(sizeof (int)*10);
for ( int k = 0 ; k < 10 ; k++) {
    PtrMas[k] = rand(); // Заполнение массива числами случайными // от 0 до 32767
// Печать элемента массива
printf("Случайное число № %d. = %d\n",k+1,PtrMas[k] );    };
free(PtrMas); // освобождение ДП
```

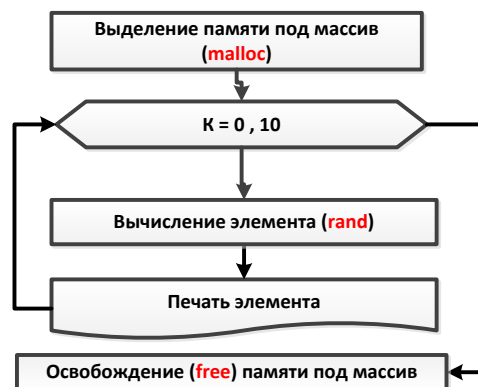
Получим в результате:

Динамический массив и Случайные числа:

Случайное число № 1. = 15724  
 Случайное число № 2. = 11478  
 Случайное число № 3. = 29358  
 Случайное число № 4. = 26962  
 Случайное число № 5. = 24464  
 Случайное число № 6. = 5705  
 Случайное число № 7. = 28145  
 Случайное число № 8. = 23281  
 Случайное число № 9. = 16827  
 Случайное число № 10. = 9961

Блок-схема примера:

Блок-схема



### 3.27. 4.8 Указатели и динамические массивы

В тестовом проекте примеров проверьте работу примеров, связанных с указателями и динамическими массивами, расположенных в теоретической части ЛР (см. выше). Проверку выполнения операций присваивания с помощью указателей необходимо произвести с помощью отладчика. Результаты необходимо продемонстрировать и преподавателю в отладчике СП.

### 3.28. 4.9 Итерационный цикл расчета значения функции на основе ряда

Следующий пример – итерационные вычисления. Формула для вычисления значения функции – вычисление суммы членов ряда имеет вид:

ОП ГУИМЦ 2024 ЛРН№3

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{n=0}^{\infty} \frac{x^n}{n!}, x \in \mathbb{C}$$

Нужно составить циклическую программу для вычисления суммы этого ряда с точностью eps. Для математического обоснования алгоритма обратитесь к учебнику по математике.

////////// цикл итерационный для функции exp(y)

#pragma warning(disable : 4996)

```
float MasF[10];
float MasX[10];
float x0=0.0F;
float delx = 0.1F;
float x = x0;
for ( int k=0 ; k < 10;k++)
{
    x = x + delx;
float sum = 1.0f; // Начальное значение для переменной суммирования
float step = x;   // Начальное значение для степени x
float fac = 1.0f; // Начальное значение для переменной текущего факториала
float Count = 1.0f;
float eps = 0.001F; // точность, с которой будет рассчитана сумма ряда.

while ( step/fac > eps ) // проверка точности вычисления
{
    sum+= step/fac; // суммирование членов ряда
    Count += 1.0f ; // n - порядковый номер члена ряда
    fac *= Count; // fac = fac * Count; - промежуточный факториал
    step = step*x; // в степени x
};
printf(" Ряд - exp = %f Функция из библиотеки = %f Аргумент x = %f\n",sum , exp( x), (x
));
MasF[k]= sum;
MasX[k]= x; };

///
printf("\n");
system(" PAUSE");
///
};
//////////
```

Результат:

```
Ряд - exp = 1.105000 Функция из библиотеки = 1.105171 Аргумент x = 0.100000
Ряд - exp = 1.221333 Функция из библиотеки = 1.221403 Аргумент x = 0.200000
Ряд - exp = 1.349500 Функция из библиотеки = 1.349859 Аргумент x = 0.300000
Ряд - exp = 1.491733 Функция из библиотеки = 1.491825 Аргумент x = 0.400000
Ряд - exp = 1.648438 Функция из библиотеки = 1.648721 Аргумент x = 0.500000
Ряд - exp = 1.821400 Функция из библиотеки = 1.822119 Аргумент x = 0.600000
Ряд - exp = 2.013572 Функция из библиотеки = 2.013753 Аргумент x = 0.700000
Ряд - exp = 2.225131 Функция из библиотеки = 2.225541 Аргумент x = 0.800000
Ряд - exp = 2.458759 Функция из библиотеки = 2.459603 Аргумент x = 0.900000
Ряд - exp = 2.718056 Функция из библиотеки = 2.718282 Аргумент x = 1.000000
```

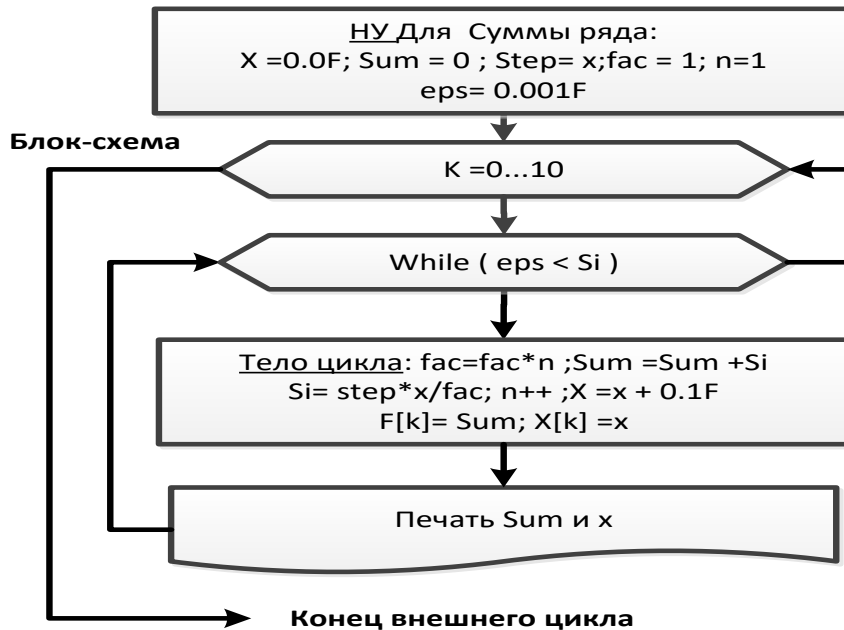
Измените вручную значение x (сейчас оно равно 0.0, шаг = 0.1) прямо в программе и проведите повторные вычисления. В контрольном задании вы должны построить цикл для разных значений x и запомнить в массиве вычисленные значения функции. Красным цветом помечено

## ОП ГУИМЦ 2024 ЛР№3

выводимое значение для библиотечной функции, а синим значение степенного ряда, вычисленное в цикле..

Для более детального знакомства с алгоритмом приближенного вычисления значения алгебраической функции смотрите разделы математики – ряды Тейлора (конкретно ряды Маклорена) в [wikipedia.org](https://wikipedia.org).

Блок-схема примера:



#### 4. 5. Контрольные задания для ЛР №3.

Следующие контрольные задания являются обязательными для выполнения в данной ЛР. Студенты эту часть выполняют после изучения теоретической части ЛР и проверки тестовых примеров основных тем ЛР.

##### 4.1. 5.1 Создание консольного проекта

Создать пустой консольный проект для выполнения ЛР № 3. Создание консольного проекта и его русификация дано в методических указаниях к ЛР №1 (см. на сайте).

##### 4.2. 5.2 Описание массивов

Описать целочисленные (**MasInt**), вещественные (**MasFloat**) и символьные (**MasChar**) одномерные массивы и инициализировать их в программе. Посмотреть содержимое массивов в отладчике и вывести на печать **5**-е элементы этих массивов ( $k = 5$ ). Размер массивов не менее 10 элементов.

Описать и инициализировать двумерный целочисленный массив **Matrica** ( $5 \times 4$ ), распечатать элемент этого массива с номерами (индексы)  $i=3, j=2$ .

##### 4.3. 5.3 Ввод и вывод массива. Вычисление суммы элементов



ОП ГУИМЦ 2024 ЛР№3

Для целочисленного массива **MasInt** (тип `int`), размерностью 5 элементов, выполнить следующие действия:

- Выполнить в отдельном цикле ввод элементов массива с клавиатуры.
- Вычислить сумму элементов этого массива в отдельном цикле
- Распечатать сумму этого массива (в отдельной строке).

В результате получим:

Введите 5 целых чисел:

[0]:1

[1]:2

[2]:3

[3]:4

[4]:5

Сумма массива MasInt = 15

Для продолжения нажмите любую клавишу . . .

#### 4.4. 5.4 Печать массива столбиком

Распечатать введенный в предыдущем пункте контрольного задания массив столбиком.

Фрагмент программы выглядит так:

В результате получим:

Элемент массива MasInt[ = 0 ] = 1

Элемент массива MasInt[ = 1 ] = 2

Элемент массива MasInt[ = 2 ] = 3

Элемент массива MasInt[ = 3 ] = 4

Элемент массива MasInt[ = 4 ] = 5

#### 4.5. 5.5 Определение минимума в массиве и его номера

Во введенном массиве, в отдельном цикле, выполняется поиск максимального элемента и запоминание его номера. Результаты поиска вывести на экран. Программу отладить и проверить на разных примерах. Составить блок-схему программы.

В результате получим:

Введите 5 целых чисел:

Элемент массива MasInt[ = 0 ] = 1

Элемент массива MasInt[ = 1 ] = 3

Элемент массива MasInt[ = 2 ] = 5

Элемент массива MasInt[ = 3 ] = 0

Элемент массива MasInt[ = 4 ] = 1

В массиве Min= 0 его номер N = 3

#### 4.6. 5.6 Инициализация массива в программе и определение отдельных сумм положительных и отрицательных элементов

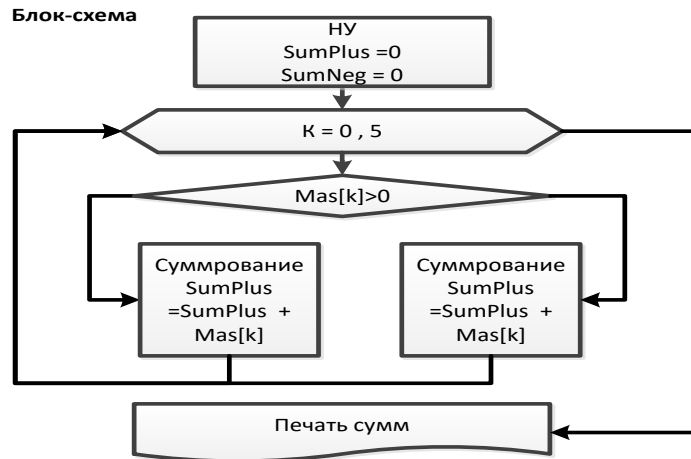
Описать массив действительных(`double`) чисел **MasD** (6 элементов) и выполнить его инициализацию при описании. Константы инициализации массива должны быть разных знаков. Построить один цикл для вычисления отдельных сумм положительных и отрицательных элементов. Результаты вычислений сумм вывести на экран.

```
double MasD[] = {3.0, -5.1, 1.0, -7.2};
```

В результате получим:

Сумма положительных = 4.000000

Сумма отрицательных = -12.300000



#### 4.7. 5.7 Использование указателя для массива

Описать указатель на тип целый. Присвоить его адрес номеру минимального элемента из предыдущего задания данного раздела. Вывести на печать (**printf**) значение этого элемента с помощью указателя.

```
// 5.6. Использование указателя для массива
int * pMin;
pMin = &MasInt[NumMin];
printf("\nПечать с указателем = %d \n", *pMin);
```

В результате получим:

Печать с указателем = 0

#### 4.8. 5.8 Вычисление и запоминание значений массива из значений вычисленного ряда.

По варианту соответствующему номеру в журнале группы, построить программу вычисления значения функции (см. таблицу). Результаты вычисления заносятся в массивы: для аргументов и для функций (**MasF**, **MasX**).

Вычислить массив **MasF (float)** по формуле специального ряда для заданной библиотечной функции для каждого значения аргумента  $x$  (по вариантам – см. ниже п.6). Результат вычисления для каждого значения аргумента запомнить в массиве – **MasF** (и показать преподавателю результат в отладчике). Значения и шаг изменения аргумента  $x$  заданы в таблице вариантов (см. ниже.). Распечатать результат в форме таблицы (номер - значение функции - значение функции из библиотеки, - значение аргумента). Таблицу оформить рамкой из звездочек (“\*”). Цикл итерационных вычислений значений рассмотрен выше (см. п.п. 4.4). Для реализации данной задачи необходим двойной цикл. Проверить полученные вычисления с помощью вычисления библиотечной функции (у нас в примере **exp**). Для вычисления значения функции из библиотеки нужно подключить заголовочный файл математической библиотеки (**math.h**):

Получим результат:

Значение ряда <b>exp</b> = 1.105000	Значение $x$ = 0.100000
Значение ряда <b>exp</b> = 1.221333	Значение $x$ = 0.200000
Значение ряда <b>exp</b> = 1.349500	Значение $x$ = 0.300000
Значение ряда <b>exp</b> = 1.491733	Значение $x$ = 0.400000
Значение ряда <b>exp</b> = 1.648438	Значение $x$ = 0.500000
Значение ряда <b>exp</b> = 1.821400	Значение $x$ = 0.600000
Значение ряда <b>exp</b> = 2.013572	Значение $x$ = 0.700000

ОП ГУИМЦ 2024 ЛРН№3

Значение ряда  $\exp = 2.225131$  Значение  $x = 0.800000$

Значение ряда  $\exp = 2.458759$  Значение  $x = 0.900000$

#### **4.9. 5.9 Заполнение и сумма в динамическом массиве через указатель.**

Создать динамический массив целого типа размером 10 элементов. Заполнить его случайными числами в диапазоне 0-100 (использовать функцию `rand`(случайное число) из библиотеки `<stdlib.h>`). Вычислить сумму элементов этого массива и распечатать результат подсчета и полученный массив.

#### **4.10. 5.10 Двумерные массивы.**

Описать и проинициализировать в программе двумерный целочисленный массив размеров  $3 \times 5$  элементов.

#### **Блок-схемы заданий**

Для всех циклических фрагментов собственных программ в отчете оформляются их блок-схемы. Желательно использовать VS Visio или MS WORD.

#### **5. 6. Контролируемые требования.**

- Описания массивов п.п 5.2
- Ввод массива п.п 5.3
- Печать массива столбиком) - п.п 5.3,4
- Сумма элементов массива п.п 5.3
- Минимум в массиве - п.п 5.5
- Инициализация и суммы положительных и отрицательных- п.п 5.6
- Указатель для минимального элемента массива и печать - п.п 5.7
- Запоминание в массиве суммы ряда и печать п.п 5.8
- Описание двумерных массивов - п.п 5.10
- Заполнение и сумма в динамическом массиве из 10-ти элементов (СМ) - п.п 5.9
- Оформление отчета по заданным требованиям.
- Выполнение дополнительных требований при наличии.

#### **6. 7. Варианты заданий для студентов ГУИМЦ.**

Варианты заданий для приближенного вычисления функции на основе ряда Маклорена приведены ниже. Номер варианта должен соответствовать номеру студента в групповом журнале.

## ОП ГУИМЦ 2024 ЛР№3

В отчет по 3-й ЛР нужно получить вставить две зависимости полученные экспериментально для своего ряда(в табличке):

1.Вычисляемый параметр: это число итераций для достижения заданной точности(eps) - N во внутреннем цикле программы.

2. Варьируемые параметры:

x - аргумент вычисления ряда при фиксированной точности (eps)

eps - точность вычисления ряда при фиксированном аргументе (x)

3. Т.е. получить зависимости:  $N=F(x)$ , при  $eps = const$   $x=var$  и

$$N= F(eps) \text{ при } x = const. \text{ eps}=var$$

Нужно подобрать такие фиксированные параметры (x или eps),

чтобы зависимости были "красивыми"(выразительными), т.е не постоянными или вырожденными. eps - точность вычисления ряда при фиксированном аргументе (x)

3. Т.е. получить зависимости:  $N=F(x)$ , при  $eps = const$  и

$$N= F(eps) \text{ при } x = const.$$

Нужно подобрать такие фиксированные параметры (x или eps),

чтобы зависимости были "красивыми"(выразительными), т.е не постоянными или вырожденными.

№ п/п	Диапазон и шаг для x	Формула для вычисления ряда для функции
1	x = 0.0 – 1.0 шаг 0.1 (atan функция из библиотеки math.h)	$\operatorname{arctg}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \dots = \sum_{n=0}^{\infty} (-1)^n x^{2n+1} / (2n+1)$
2	x = 0.1 – 2.0 шаг 0.3 (cosh)	$\cosh x = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + \frac{x^{2n}}{(2n)!} + \dots$
3	x = 1.0 – 5.0 шаг 0.4 (sin)	$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}, x \in \mathbb{C}$
4	x = 0.0 – 1.0 шаг 0.1 (cos)	$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}, x \in \mathbb{C}$
5	x = -0.0 – (-1.0) шаг -0.1 (asin)	$\arcsin x = x + \frac{x^3}{6} + \frac{3x^5}{40} + \dots = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1}$
6	x = 1.0 – 5.0 шаг 0.4 (atan)	$\operatorname{arctg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \dots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1}$
7	x = 0.1 – 2.0 шаг 0.3 (sinh)	$\operatorname{sh}(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots = \sum_{n=0}^{\infty} \frac{1}{(2n+1)!} x^{2n+1}, x \in \mathbb{C}$
8	x = -0.0 – (-1.0) шаг -0.1 (sqrt) – корень	$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{x^3}{16} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(1-2n)n! 2^{2n}} x^n,$
9	x = 0.0 – 1.0 шаг 0.1 (exp)	$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{n=0}^{\infty} \frac{x^n}{n!}, x \in \mathbb{C}$
10	ln(x+1) x = 0.0 – 1.0 (log)	$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots = \sum_{n=0}^{\infty} (-1)^n x^{n+1} / (n+1)$
11	exp(x) x=0.0-1.0 шаг 0.3 (exp)	$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{n=0}^{\infty} \frac{x^n}{n!}, x \in \mathbb{C}$

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

Для продвинутых студентов, по желанию, в дополнение к основному заданию, можно построить программу с дополнительными требованиями.

#### 7.1. 8.1 Ввод и вывод двумерного массива. Вычисление суммы его элементов.

Выполняется ввод двумерного действительного массива (3\*4) построчно (**double**). Первоначально вводятся значения чисел массива с подсказкой номера строки и столбца. Далее вычисляется сумма всех элементов и распечатывается таблично массив и результат расчета. Сумма выводится в отдельной строке.

#### 7.2. 8.2 Определение максимума в двумерном массиве и координат максимума

Выполняется ввод двумерного массива. Наполнение массива задается с помощью инициализации в программе. Выполняется поиск максимального/минимального элементов в массиве. Массив печатается в виде таблицы. Значение максимума/минимума и его координат выводятся в отдельных строках.

#### 7.3. 8.3 Инициализация двумерного массива

Продемонстрировать возможности для инициализации двумерного массива действительного типа. Массив вывести на экран в виде таблицы.

#### 7.4. 8.4 Вычисления значений полинома

Сделать программу для вычисления полинома общего вида:

$$y = a_1x^n + a_2x^{n-1} + \dots + a_nx + a_{n+1}$$

Значения коэффициентов задаются в отдельном массиве **AMas**. Для вычисления суммы ряда использовать схему Горнера. Степень полинома равна 10-ти. Сделать алгоритм вычисления максимально эффективным. Значения  $x$  задать в диапазоне от 0 до 10 с шагом 0.1. Результат вычислений вывести в табличной форме.

#### 7.5. 8.5 Описание и инициализация двумерных массивов и умножение матриц

Описать три массива:  $A$  (1\*3),  $B$  (1\*4) и  $C$  (3\*4). Массивы  $A$  и  $B$  инициализировать в программе. Вычислить массив  $C$  – матрицу, как произведение массивов матриц  $A$  и  $B$ .

$C = A * B$ . (Это задание для студентов, изучивших курс матричной алгебры). Для поиска формулы расчета использовать справочники или Интернет.

#### 7.6. 8.6 Доказательство расположения массива.

Доказать в программе способ расположения элементов в двумерном массиве (с помощью печати массива через указатель).

#### 7.7. 8.7 Блок-схемы заданий

Для всех фрагментов программ оформляются их блок-схемы и размещаются в отчете ЛР.

**Примечание:** Выполнение программы с дополнительными требованиями отображается в журнале, в отчете по ЛР и учитывается при подведении результатов работы в семестре и на экзамене.

### **8. 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.

### **9. 10. Контрольные вопросы по ЛР.**

1. Что такое массив в ЯП?
2. Что такое указатель в ЯП?
3. Для чего нужны массивы в ЯП?
4. Для чего нужны указатели в ЯП?
5. Как может задаваться размерность массива, покажите в вашей программе?
6. Что такое инициализация массива?
7. Как инициализировать двумерный массив?
8. Может ли быть при инициализации не задан размер массива?
9. Что такое динамические массивы?
10. Как описываются указатели?
11. Какие новые операции для указателей вы знаете?
12. Что такое операция именованная?
13. Что такое операция разыменованная?
14. Можно ли задавать при описании указателя несколько звездочек?
15. Можно ли приравнять указатель и имя массива?
16. Как обратиться к элементу массива с помощью указателя?
17. Могут ли быть в массиве заданы элементы разного типа?
18. Как вычислить размер массива при выполнении программы?
19. В чем суть численных алгоритмов вычисления функции в виде ряда?
20. Какие элементы блок схем вы знаете?
21. Какие преимущества дают проекты в VS?
22. Как описать указатель на массив и его вычислить?
23. Как описать и использовать указатель на указатель?
24. Что такое динамическая память?
25. Какие функции для работы с динамической памятью вы знаете?
26. Как освободить динамическую память.?
27. Как можно выполнить ввод массива?
28. Как описать и работать с многомерными массивами?
29. Какие операции используются для работы с указателями?
30. Как можно задать и вычислить размер массива?

**10. 11 Литература.****Основная литература**

1. Список литературы, доступные книги и необходимые пособия для ЛР ОП размещены на сайте [www.sergebolshakov.ru](http://www.sergebolshakov.ru) на страничке “2-й к СУЦ”. Пароль для доступа можно взять у преподавателя или старосты группы.
2. Керниган Б., Ритчи Д. К Язык программирования С, 2-е издание: Пер. с англ. – М. : Издательский дом “Вильямс”, 2009. – 304с.: ил. – Пар. Тит. англ.
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](http://www.sergebolshakov.ru)) – см. кнопку в конце каждого раздела сайта!!!
8. Керниган Б., Ритчи Д. К36 Язык программирования Си.\Пер. с англ., 3-е изд., испр. - СПб.: "Невский Диалект", 2001. - 352 с.: ил.
9. Другие методические материалы по дисциплине с сайта [www.sergebolshakov.ru](http://www.sergebolshakov.ru).
10. Конспекты лекций по дисциплине “Основы программирования”.
11. Подбельский В.В. Язык Си++: Учебное пособие. – М.: Финансы и статистика, 2003.
12. 5. Подбельский В.В. Стандартный Си++: Учебное пособие. – М.: Финансы и статистика, 2008.
13. Г. Шилдт “С++ Базовый курс”: Пер. с англ.- М., Издательский дом “Вильямс”, 2011 г. – 672с
14. Г. Шилдт “С++ Руководство для начинающих” : Пер. с англ. - М., Издательский дом “Вильямс”, 2005 г. – 672с
15. Г. Шилдт “Полный справочник по С++”: Пер. с англ.- М., Издательский дом “Вильямс”, 2006 г. – 800с
16. Бьерн Страуструп "Язык программирования С++"- М., Бином, 2010 г.