

**Методические указания к лабораторной работе № 2 по курсу
ОСНОВЫ ПРОГРАММИРОВАНИЯ
ГУИМЦ**

**"Операторы ветвления и циклов в программах"
(4 часа)**

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
1. 1. Цель лабораторной работы №2 по дисциплине ОП(Основы программирования) - СУЦ	4
2. 2. Порядок выполнения лабораторной работы	4
3. 3. Основные понятия.....	4
3.1. 3.1 Разветвляющиеся вычислительные процессы.	4
3.2. 3.2 Безусловный переход (goto)	5
3.3. 3.3 Ветвление и операторы ветвления (if - else)	5
3.4. 3.4 Условная операция/выражение	7
3.5. 3.5 Переключатели (switch - case)	7
3.6. 3.6 Циклы (for, while, do)	8
3.7. 3.7 Отладка программ.	11
3.8. 3.8 Модули	12
3.9. 3.9 Понятие о блок-схемах	13
3.10. 3.10 Библиотеки функций.....	14
4. 4. Примеры программ с ветвлением и циклами.	14
4.1. 4.1 Пример поиска максимального значения из трех переменных (А , В, С)	15
4.2. 4.2 Решение квадратного уравнения.....	15
4.3. 4.3 Использование условного оператора (if) для организации цикла.....	17
4.4. 4.4 Использование переключателя и операторов цикла	18
4.5. 4.5 Цикл табуляции функции на основе if.....	18
4.6. 4.6 Сумма четных чисел при вводе данных	19
4.7. 4.7 Вывод на печать заданного при вводе количества чисел Фибоначи	20
4.8. 4.8 Вывод на печать заданного при вводе количества простых чисел	21
5. 5. Обязательные контрольные задания ЛР №2.....	21
5.1. 5.1 Создание консольного проекта	22
5.2. 5.2 Примеры из МУ для ЛР №2	22
5.3. 5.3 Определение минимального значения.....	22
5.4. 5.4 Использование переключателя для решения уравнения.....	22
5.5. 5.5 Цикл вычисления значений математических функции по варианту (оператор if)	22
5.6. 5.6 Цикл вычисления значений функции по варианту операторы (for, do, while).....	22
6. 6. Варианты заданий для студентов СУЦ.	23
7. 7. Дополнительные требования для сильных студентов СУЦ (д.т.).....	24
7.1. 7.1 Определение максимального значения	24
7.2. 7.2 Вычисления с запоминанием значений	24
7.3. 7.3 Вычисления с запоминанием значений и вводом аргументов	24
7.4. 7.4 Вычисления минимума в массиве.....	24
8. 8. Демонстрация, защита ЛР и отчет по ЛР.....	25

9. 9. Контрольные вопросы по ЛР.	25
10. 10. Литература.	25

1. 1. Цель лабораторной работы №2 по дисциплине ОП - ГУИМЦ

Целью данной ЛР по дисциплине ОП является получение начальных знаний и умений для программирования разветвляющихся программ на языке программирования СИ. Студенты используют консольные проекты и отлаживают программы в среде программирования MS VS 2005/2008/2010. Студенты знакомятся с операторами ветвления (условными и циклическими), 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 циклическими и условными операторами на языке программирования СИ.

3.1. 3.1 Разветвляющиеся вычислительные процессы.

Традиционно операторы выполняются последовательно (машина Тьюринга). Однако реально построить более или менее сложную программу, имеющую линейную последовательность выполняемых операторов невозможно. Для этого, помимо операторов выполняющих непосредственные вычисления (напомним – операторы присваивания значений) в языках программирования предусматриваются операторы управления или, более точно, операторы, управляющие последовательностью выполнения других операторов программы.

Возможные варианты изменения последовательности выполнения операторов следующие, они обусловлены реальными потребностями программирования, а именно:

- Необходимостью безусловного перехода к выполнению другого оператора, например завершающего программу (безусловный переход на метку).
- Необходимость выбора альтернативных путей выполнения программы (условный оператор и оператор переключатель выполнения групп операторов).
- Необходимость многократного повторения последовательности операторов (операторы циклического повторения).
- Вызов повторяющейся последовательности операторов, настраиваемых на заранее выделенные параметры (вызов процедур и функций).

Рассмотрим применительно к языку СИ предусмотренные в нем операторы управления.

3.2. 3.2 Безусловный переход (goto)

Для безусловного перехода управления к другому оператору программы, не являющимся следующим по порядку, используется оператор **goto**. Место программы, в которое осуществляется передача управления, задается специальной меткой. Метка задается именем (идентификатором) и размещается перед оператором, к которому мы хотим перейти. Признаком метки является двоеточие, которое без пробела размещается за меткой. Формально это выглядит так:

```
goto <метка>;
.....
<метка>: <оператор>;
```

Ниже приведен фрагмент программы, в котором используются метки (Lab0, Lab1) и операторы безусловной передачи управления **goto**. Оператор передачи управления на метку Lab0 закомментирован, так как при его использовании возникает бесконечный цикл – выполнение программы никогда не остановится.

```
// переход и метка
printf("Начало программы!!\n");
Lab0:
goto Lab1;
printf("Никогда не печатается!!!\n");
Lab1:
// goto Lab0; // Бесконечный цикл – закомментировано!
printf("Переходим сюда!!\n");
```

Операторы безусловной передачи управления используются в программах на СИ/СИ++ только в исключительном случае. Во-первых, доказано, что любой алгоритм можно реализовать без использования этого оператора, а, во-вторых, его применение может приводить к сложным для поиска ошибкам. В наших примерах, для организации циклов мы продемонстрирует использование этого оператора.

3.3. 3.3 Ветвление и операторы ветвления (if - else)

Часто в процессе программных вычислений, в зависимости от логических условий, определяемых переменными программы, необходимо выполнять либо одни, либо другие действия. Характерным примером может служить программа, для вычисления корней квадратного уравнения. В зависимости от значения дискриминанта мы получаем либо действительные, либо комплексные значения корней уравнения (пример такой программы мы рассмотрим ниже).

Условный оператор (if - else) позволяет сделать проверку и обеспечивает ветвление в программе. В условном операторе проверяется логическое условие и, в зависимости от результата выполняется либо одна группа операторов (составной оператор в общем случае), либо другая группа операторов (составных операторов). Формализовано это может быть записано так:

```
if (<логическое условие>)
{ <составной оператор, выполняемый при истинности условия>
[ else [ if ]
{ <составной оператор, выполняемый при ложности условия > } ]
;
```

Вторая часть условного оператора необязательна, в этом случае в программе выполняется составной оператор при истинности условия, в противном случае никаких действий просто не производится. Пример условного оператора для сравнения двух переменных, в котором по результатам сравнения выводится текст о том, какая из переменных больше:

```
// условный оператор (максимум из двух переменных)
int a = 5;
int b = 3;
if ( a > b ) // Простое условие
    printf(" a>b !\n");
else
    printf (" a<=b !\n");
```

Условные операторы могут быть вложены, внутри одного оператора, в теле составного оператора размещаются другие. Это показано на примере, в котором, после else, вставляется новый условный оператор:

```
if ( a > b )
    printf(" a>b !\n");
else
{
    if ( a < b ) // вложенный условный оператор
        printf (" a<b !\n");
    else
        printf (" a=b !\n");
};
```

Другой пример условного оператора со сложным условием:

```
if ( i > 5 && Flag ) // проверка условия
{ iMas[i] = 0; Flag = false;} // Старшим элементам массива присвоено значение 0
else
{ iMas[i] = iMasB[i]; Flag = true;}; // Младшим элементам массива - iMasB[i]
```

3.4. 3.4 Условная операция/выражение

Условное выражение/ условная операция – используется для альтернативных вычислений.

Конструкция условного выражения (правило записи):

<условное выражение> :=(<логическое условие>)?<Выражение 1>:<выражение 2>

Смысл условной операции: Сначала вычисляется <логическое условие> и, если оно истинно, то в операции участвует <Выражение 1> иначе <Выражение 2> .

<условие> - условное выражение (вычисление **true** или **false**).

Примеры условного выражения и операции:

// Примеры использования условной операции

int x =5 , y =3 , z =0;

// Использование условной операции в операторе присваивания

z = ((y > x)?y:x); // Вычисление z = max{x,y}

// В вызове функции: Печать максимального из x и y

**printf("Макс.из x = %d и y=%d равно-> %d\n" ,
x , y , ((y > x)? y : x));**

Результат применения условного выражения:

Макс.из x = 5 и y=3 равно-> 5

3.5. 3.5 Переключатели (switch - case)

Переключатели (**switch - case**) позволяют сделать выбор из множества альтернатив. Здесь для краткости мы их рассматриваем очень кратко. С ними более подробно можно познакомиться в рекомендованной литературе. Формально конструкция переключателя имеет вид:

```
switch (<выражение>) {
case <конст-выр 1>: <Составной оператор 1>
case <конст-выр 1>: <Составной оператор 1>
default: <Составной оператор K >
}
```

Кратко поясним следующее: в заголовке оператора (**switch**) проверяется целочисленное выражение (у нас просто - **num**); в зависимости от числа выполняется сравнение с константой указанной после частью выбора (**case**); если значения совпадают, то выполняется группа операторов после **case**. Далее последовательно выполняются все операторы, которые расположены в данном переключателе (вне зависимости от сравнений **case**), если такое выполнение не прерывается с помощью оператора **break**. Если такой оператор (**break**) встретился в тексте, то далее выполняется оператор, который следует за переключателем. Если совпадений с константами **case** вообще не было, то при наличии выполняется группа операторов, которая следует за ключевым словом **default**, и проверки прекращаются. Ниже приводится пример переключателя, в котором значение переменной **num** проверяется последовательно с константами 1, 2, 3.

```
// Переключатель
int num = 2;
switch ( num)
{
    case 1:
        printf("Выбор 1!\n");
        break;
    case 2:
        printf("Выбор 2!\n");
        case 3:
            printf("Выбор 3!\n");
            // break
            break;
    default:
        printf("Выбор по умолчанию!!\n");
};
```

Результат работы данного текста такой (проанализируйте, почему так):

Выбор 2!

Выбор 3!

Если для сравнения задаются символы (у нас - 'Ж'):

```
// Переключатель для символов
char csimb = 'Ж';
switch ( csimb)
{
    case 'I':
        printf("Выбор I!\n");
        break;
    case 'Ж':
        printf("Выбор Ж!\n");
    case 'N':
        printf("Выбор N!\n");
        // break
        break;
    default:
        printf("Выбор по умолчанию!!\n");
};
```

То получим:

Выбор Ж!

Выбор N!

Для продолжения нажмите любую клавишу . . .

В методическом пособии [6] в разделе 8 посмотрите, пожалуйста, как оформляется в блок-схемах оператор переключатель.

3.6. 3.6 Циклы (for, while, do)

Циклы – это фрагменты программы, которые для достижения результата нужно повторять многократно. Циклы содержат три основных элемента:

- Начальные условия цикла, выполняемые однократно для начала цикла
- Тело цикла – повторяющиеся операторы;

- Условие, проверяющее завершение или продолжение циклических повторений.

В зависимости от того как в программе записаны эти элементы, в СИ предлагаются операторы цикла различного вида: повторить заданное число раз (**for**), повторить пока истинно условие (**while**) и выполнить, проверив условие продолжения(**do - while**). Во всех случаях начальные условия цикла могут быть заданы операторами предварительно. Условие продолжения записывается в самих операторах цикла. Тело цикла задается в виде составного оператора, который может включать и другие операторы цикла и директивы описания переменных. Такие циклы называются вложенными. Приведем примеры различных операторов цикла.

В операторе цикла **for** для управления циклом в его заголовке задаются три составляющие, разделенные знаком “;” – точка с запятой: задание начального значения переменной управления циклом (может даже включаться ее описание); проверка условия продолжения цикла и группа операторов, выполняемых после завершения каждого шага цикла перед проверкой условия. Приведенный ниже цикл (его тело – оператор печати) выполняется 5 раз для *i* от 1 до 5 включительно с шагом 1 (*i*++). Далее выполняется следующий оператор, расположенный после тела цикла.

```
// Цикл for
for (int i = 1 ; i <= 5 ; i++ )
{
    printf("Шаг цикла(for) - %d\n" , i);
};
// Или
for (int i = 1 ; i <= 5 ; i++, k = 10 , j++ ) // несколько выражений
{
    printf("Шаг цикла(for) - %d\n" , i); // Тело цикла };

```

В операторе цикла **while** сначала проверяется условие продолжения цикла, указанное в заголовке. Если условие истинно, то тело цикла выполняется, а затем снова проверяется условие продолжения. Когда устанавливается факт ложности условия, повторы тела цикла прекращаются. Приведенный ниже цикл выполняется 5 раз для *k* от 0 до 4. (Заметьте, последнее значение *k* равно 5-ти). Далее выполняется следующий оператор, расположенный после тела цикла.

```
// Цикл while
int k = 0;
while ( k < 5)
{
    printf("Шаг цикла (while) - %d\n" , k);
    k++;
};

```

В операторе цикла **do** сначала однократно (всегда) выполняется тело цикла, а затем проверяется условие продолжения цикла (**while**), указанное после тела. Если условие истинно, то тело цикла снова выполняется, а затем снова проверяется условие продолжения. Когда устанавливается факт ложности условия, повторы тела цикла прекращаются. Далее выполняется следующий оператор, расположенный после тела цикла. Приведенный ниже цикл выполняется 2 раза для *i* от 0 до

2, шагом 2. (Заметьте, последнее значение i равно 4-ти). Далее выполняется следующий оператор, расположенный после тела цикла.

```
// Циклы do
int i = 0;
do
{
    printf("Шаг цикла(do) - %d\n", i);
    i++; i++;
} while ( i < 4);
```

Оператор **break**, также как и в переключателе прерывает выполнение цикла любого типа. После его выполнения в теле цикла циклические повторения прекращаются, управление передается следующему оператору, расположенному за циклом.

```
// Цикл for с условной остановкой на втором шаге
for (int i = 1 ; i <= 5 ; i++ )
{
    printf("Шаг цикла(break) - %d\n", i);
    if ( i == 2) break;
};
```

Оператор **continue** прерывает выполнение только текущего шага цикла. Далее цикл продолжается так, как задано по начальному условию. В примере, расположенном ниже часть 2 цикла не выполняется при значении индекса i равного 2-м.

```
// Цикл for с условным пропуском части операторов на втором шаге (
continue)
for (int i = 1 ; i <= 5 ; i++ )
{
    printf("Шаг цикла(break) - %d", i);
    if ( i == 2) { printf("Пропуск %d!!\n", i); continue; };
    printf("Тело - часть 2!!\n");
};
```

Рассмотрим и другие примеры использования операторов циклов (**for**, **while** и **do**).

Для цикла **for** (суммирование массива):

```
Summ = 0; // начальное условие
for (int i = 0 ; i < MAX; i++ ) // начальное условие и проверка
    продолжения
{
    Summ = Summ + iMas[i]; // тело цикла
    ... // Другие операторы тела цикла
};
```

Для цикла **while**, тоже вычисление суммы массива:

```
Summ = 0; // начальные условия
int i = 0; // начальные условия
```

```

while (i < MAX ) // проверка продолжения
{
    Summ = Summ + iMas[i]; // тело цикла
    i++;
    ... // Другие операторы тела цикла
};

```

Для цикла **do – while** (сумма в массиве):

```

Summ = 0; // начальные условия
int i =0; // начальные условия
do {
    Summ = Summ + iMas[i]; // тело цикла
    i++;
    ... // Другие операторы тела цикла
} while (i < MAX ); // проверка продолжения

```

Большинство ошибок в циклической программе связано с неверным заданием условий проверки продолжения и завершения циклов, а также задании начальных условий цикла. В этом случае часто программа может выполняться сколь угодно долго и говорят, что программа зациклилась (заметьте, что данное слово уже вошло также в обиход обычной речи!).

3.7. 3.7 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!

3.8. 3.8 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(){
```

```

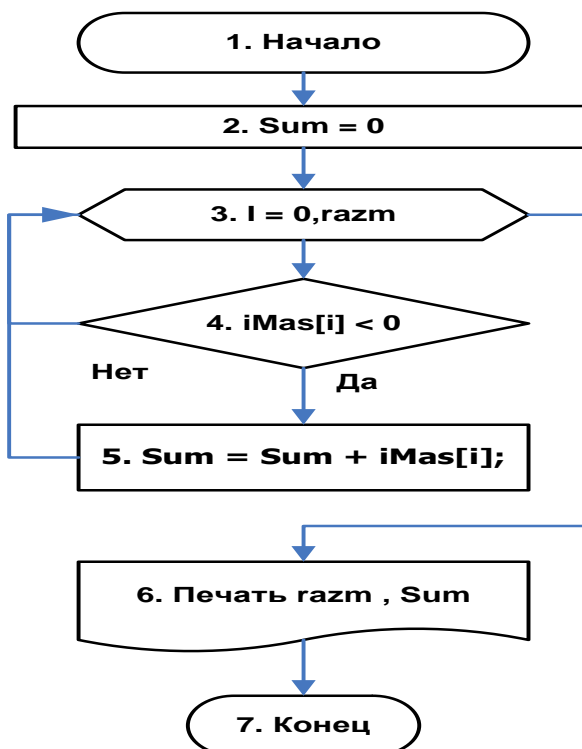
...
i = 5; // Использование внешних данных
S = Summ(2,2); // Вызов внешней функции
...
}

```

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`.

3.9. 3.9 Понятие о блок-схемах

Для описания алгоритмов используются различные способы. Наиболее наглядным и простым является технология блок-схем. В лабораторных работах вам необходимо в совершенстве освоить эту технологию. В ЛР №2 фрагменты программ циклического вида и с ветвлением нужно оформить в виде блок-схемы. Блок-схемы позволяют не только логично и без ошибок построить программу, но и проверить ее работу в процессе отладки и проверки. Технология построения блок-схем подробно описана в методическом пособии к лабораторным работам по курсу[5] в разделе 8. Для графического построения блок-схем очень удобно использовать программный продукт MS



Visio. Пример блок-схемы для вычисления суммы положительных элементов массива приведен ниже.

3.10. 3.10 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.

3.11. 3.11. 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 программ с ветвлением и циклами.

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 Пример поиска максимального значения из трех переменных (А , В, С)

Задавая вручную различные значения переменных **А, В, С** нужно проверить работы алгоритма по всем веткам условий (четыре варианта).

// условный переход

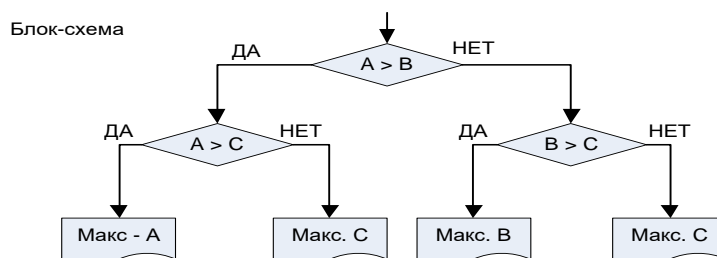
int A = 4 , B = 4 , C =1;

```
if ( A > B )
{
    if ( A > C )
        printf("Максимально (A) - %d\n" , A);
    else
        printf("Максимально (C) - %d\n" , C); }
else
{
    if ( B > C )
        printf("Максимально (B) - %d\n" , B);
    else
        printf("Максимально (C) - %d\n" , C); };
```

Результат выполнения:

Максимально (В) – 4

Блок-схема фрагмента выглядит так:



4.2. 4.2 Решение квадратного уравнения

Задавая вручную различные значения переменных **a, b, c** нужно проверить работы алгоритма по всем веткам условий (два варианта). Уравнение в общем виде имеет вид. Его решение вы можете посмотреть в любом справочнике по математике. Общий вид квадратного уравнения:

$$ax^2+bx+c=0$$

Программа для вычисления корней уравнения (Напомним: корни могут быть действительными и комплексными, см. любой справочник по математике).

```
float a,b,z;
float c,d,e,f,x1,x2;
printf("\nВведите a b и c:\n");
scanf ("%5f%5f%5f",&a,&b,&c);
printf("\nДля a = %4.2f   b = %4.2f   c= %4.2f \n" , a, b ,c );
// Промежуточные вычисления для упрощения
z=2.0f*a;
e=-b/z;
d=b*b-4*a*c; // детерминант (дискриминант)
f=sqrt(fabs(d))/z; // функции библиотеки
// Проверка корней
if(d>=0)
{ x1=e+f;
  x2=e-f;
  printf("Корни уравнения действительные\n");
  printf("\n x1= %4.1f",x1);
  printf("\n x2= %4.1f\n",x2);  }
else
{ printf("Корни уравнения комплексные\n");
  printf("\nВещественная часть корня: e= %4.1f",e);
  printf("\nМнимая часть корня: f= %4.1f\n",f); };
```

Результат выполнения:

Введите a b и c:

5

8

2

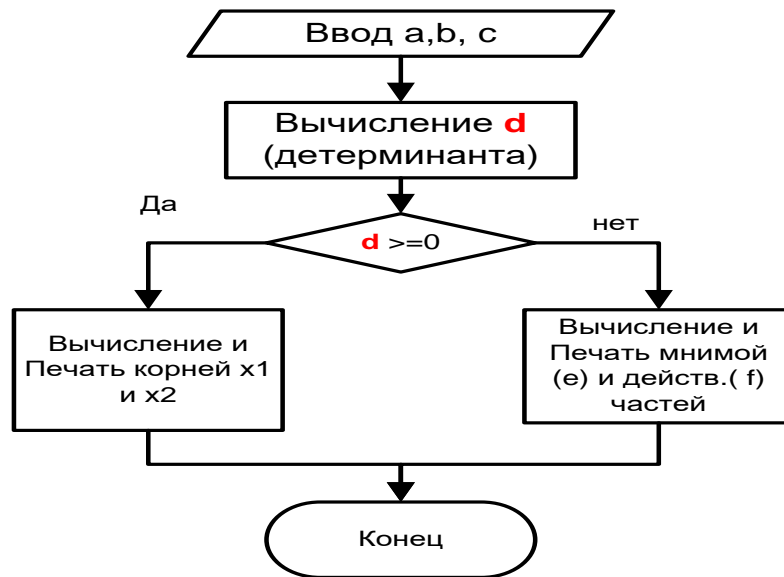
Для a = 5.00 b = 8.00 c= 2.00

Корни уравнения действительные

x1= -0.3

x2= -1.3

Блок-схема программы:



4.3. 4.3 Использование условного оператора (if) для организации цикла

Для организации цикла может быть использован и условный оператор `if`. Как это делается показано в следующем фрагменте текста для циклического вычисления суммы арифметической прогрессии.

```

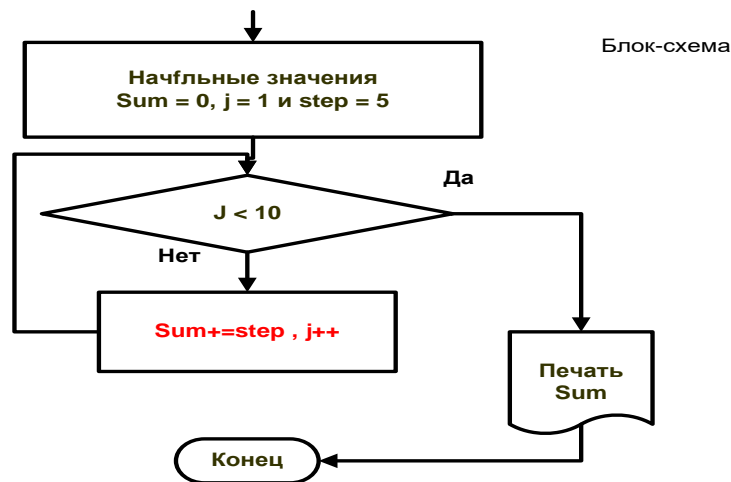
int j = 1; // Начальные условия цикла
int Sum = 0;
int step = 5;
M1:
if ( j < 10) // Проверка продолжения цикла
{
    Sum +=j*step; // Тело цикла
    j++;          // Тело цикла
goto M1;         // Тело цикла
};

printf ("\nЗначение Sum = %d для j =%d\n" , Sum, j);

```

Результат:
Значение Sum = 225 для j =10

Блок-схема программы суммирования:



4.4. 4.4 Использование переключателя и операторов цикла

Проверьте использование оператора переключателя из теоретической части данной ЛР. Для этого в ручную в режиме отладки задавайте разные значения для переменной num (1,2,3)

Проверить выполнение всех примеров из других разделов методических указаний ЛР №2 для условных и безусловных операторов (**goto**, **if**, **if-else**), и операторов цикла (**for**, **while** и **do**) с вариантами **break** и **continue**.

Дополнительные примеры для освоения темы смотрите в разделах расположенных выше и ниже. Желательно при изучении материала создать новый консольный проект и копировать в него программы из текста этого документа.

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

4.5. 4.5 Цикл табуляции функции на основе if

Для вычисления функции (z) для разных аргументов нужно организовать цикл с оператором **goto**. Пусть нужно вычислить функцию:

$$z = 5 \sin(0.5bx^2)/x.$$

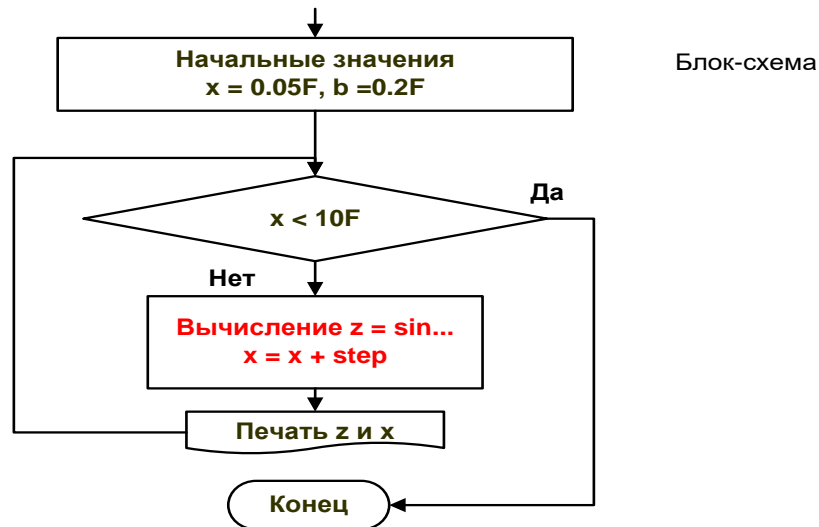
Для вычисления (табуляции) этой функции можно использовать следующий фрагмент программы:

```

// Начальные условия цикла
float x= 0.05F; // начальное значение (!=0)
int b = 2; int a = 1; // здесь a и b задаем при инициализации
float Step = 1.0F, z;
// Цикл табуляции функции
СИКЛ:
if ( x <= 10.f ) { // Проверка завершения
    z= (float) (5.0*sin( (0.5F*b*pow(x,2)))/x); // Вычисление z row - функция возведения в
    степень
    // z=cos(2*a); // Ошибка a - int
    // z=cos(2.0*a); // использование действительной константы
    // z=cos((double)2*a); // использование cast- преобразований
    // Вывод строки
    printf("\nФункция z =%7.3f для x= %4.2f", z, x);
    x += Step; // изменение x на шаг step
}
  
```

`goto CIKL; // Продолжение цикла`

`};` Блок-схема фрагмента программы имеет следующий вид:



Результат:

```

Функция  z =  0.250 для x= 0.05
Функция  z =  4.249 для x= 1.05
Функция  z = -2.129 для x= 2.05
Функция  z =  0.200 для x= 3.05
Функция  z = -0.790 для x= 4.05
Функция  z =  0.358 для x= 5.05
Функция  z = -0.735 для x= 6.05
Функция  z = -0.379 для x= 7.05
Функция  z =  0.572 для x= 8.05
Функция  z =  0.121 для x= 9.05
  
```

4.6. 4.6 Сумма четных чисел при вводе данных

С клавиатуры вводиться максимальное число циклов, каждый из которых используется для ввода одного числа. Если число четное то оно суммируется, если введен 0, то цикл заканчивается.

// 4.5. Сумма четных при вводе первых десяти чисел

```

int SummEven = 0; // Сумма четных чисел
int CicleCount= 0; // счетчик циклов для ввода
int CiclMax=0; // Максимальное число циклов
int iWork = 1; // Введенное значение числа
// Ввод числа циклов
printf("Введите число циклов ввода:");
scanf ("%d", &CiclMax);
while (CicleCount!=CiclMax && iWork!=0 )
{
    printf("Введите новое число (осталось циклов - %d):", (CiclMax - CicleCount) );
    scanf ("%d", &iWork);
    if (iWork%2 == 0) // Проверка четности iWork
        SummEven+=iWork;
    CicleCount++;
    if (CicleCount == CiclMax) break;
};
printf("Сумма первых %d четных чисел = %d \n при , макс. ввода = %d \n", CicleCount ,
SummEven , CiclMax );
  
```

```
/////
```

Получим при вводе трех чисел (2,3,5):

```
Введите число циклов ввода:3
Введите новое число (осталось циклов - 3):2
Введите новое число (осталось циклов - 2):3
Введите новое число (осталось циклов - 1):5
Сумма первых 3 четных чисел = 2
при , макс. ввода = 3
```

4.7. 4.7 Вывод на печать заданного при вводе количества чисел Фибоначи

Каждое новое число является суммой двух предыдущих чисел Фибоначи (см. Wikipedia.ru). Нужно построить программу для вывода заданного числа этих чисел на экран, начиная с первого числа.

$F_n = F_{n-1} + F_{n-2}$; Причем n – натуральное целое число ($0 < n$), $F_0 = 0$ и $F_1 = 1$

Пример такой программы показан ниже:

```
//////////
//4.7. Вывод на печать заданного количества чисел Фибоначи
int FibCount = 0;
// Fn = Fn-1 + Fn-2; Причем n – натуральное целое число (0 - n), F0 = 0 и F1 = 1
printf("Введите число выводимых чисел Фибоначи (n):");
scanf ("%d", &FibCount); // Ввод количества выводимых чисел
printf("\nВыводим %d чисел Фибоначи!\n:" ,FibCount );
// Первые два числа
int F0=0 , F1=1, FN;
printf("Число N - %d = %d \n" , 0 , F0 );
printf("Число N - %d = %d \n" , 1 , F1 );
// Новое число
FN = F0 + F1;
// Цикл вывода чисел
for (int i =2 ; i <=FibCount ; i++ )
{
    printf("Число N - %d = %d \n" , i , FN );
    F0 = F1 ;
    F1 = FN;
    FN = F0 + F1; // Вычисляем новое число
};
system(" PAUSE");
return;
};
```

Получим при выводе 10 чисел Фибоначи:

```
Введите число выводимых чисел Фибоначи (n):10
Выводим 10 чисел Фибоначи!
Число N - 0 = 0
Число N - 1 = 1
Число N - 2 = 1
Число N - 3 = 2
Число N - 4 = 3
Число N - 5 = 5
Число N - 6 = 8
Число N - 7 = 13
Число N - 8 = 21
Число N - 9 = 34
Число N - 10 = 55
```

Для продолжения нажмите любую клавишу . . .

4.8. 4.8 Вывод на печать заданного при вводе количества простых чисел

Простые числа это такие числа, которые делятся нацело на 1 и на себя (1,2,3, 5, 7, 11, ...). Другие числа называются составными (4, 6, 8, ...).

Нужно построить программу, которая выводит на экран заданное число (при вводе) простых чисел. Такая программа показана ниже.

```
// 4.8 Простые числа заданное число m
int m = 0;
printf("Введите число выводимых простых чисел (m):");
scanf ("%d", &m); // Ввод количества выводимых чисел
printf ("%d - первых простых чисел \n", m);
int P = 37;
for (int k = 2 ; k <= m; )
{
    P = k;
    // Проверка на простое
    int FlagP = 0; // Флаг наличия делителя – флаг простого числа
    printf ("\n%d.", k );
    for ( int j =2 ; j <= P ; j++ )
    {
        if ( ( P % j == 0) && ( P!=j) ) FlagP = 1; // проверить ///
    };
    if (FlagP == 0)
    { printf ("P = %d - простое!", P); k++; }
    else
    { printf ("P = %d - составное!", P); k++; };
};
printf ("\n" );
system(" PAUSE");
};
Введите число выводимых простых чисел (m):10
10 - первых простых чисел
Получим при выводе 10 простых чисел:
2.P = 2 - простое!
3.P = 3 - простое!
4.P = 4 - составное!
5.P = 5 - простое!
6.P = 6 - составное!
7.P = 7 - простое!
8.P = 8 - составное!
9.P = 9 - составное!
10.P = 10 - составное!
Для продолжения нажмите любую клавишу . . .
```

5. 5 Обязательные контрольные задания ЛР №2.

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.

5.1. 5.1 Создание консольного проекта

Создать пустой консольный проект для выполнения ЛР № 2. Создание консольного проекта и его русификация дано в методических указаниях к ЛР №1 (см. на сайте).

5.2. 5.2 Примеры из МУ для ЛР №2

Сделать все примеры из теоретической части ЛР №2. (Раздел 4). Для этого можно в свой проект копировать тексты программ непосредственно из данных методических указаний. Желательно иметь работающий и русифицированный свой проект для этих целей.

5.3. 5.3 Определение минимального значения

Написать программу для определения минимального значения из трех целых переменных (тип **int**): А, В, С. Переменные задаются в программе с помощью оператора присваивания (без ввода). Результаты сравнения и исходные значения переменных вывести на экран (функцией **printf**). Построить блок-схему программы.

5.4. 5.4 Использование переключателя для решения уравнения

Переделать программу для вычисления корней квадратного уравнения (см. выше) с использованием переключателя (**switch**) для **d** (детерминанта квадратного уравнения). Примечание: для этого нужно добавить переменную целого типа и вычислить ее по условию значения детерминанта **d** (1 или 2). Построить блок-схему программы. Коэффициенты уравнения (**a,b,c**) задать в программе с помощью присваивания значений.

5.5. 5.5 Цикл вычисления значений математических функции по своему варианту (оператор if)

Составить циклическую программу на основе оператора if для вычисления и вывода значений функции по заданному варианту (см. ниже). Формула для вычисления заданной функции **z** и заданного диапазона значений аргументов **x** задается вариантом (см. раздел 6 ниже). Параметр **a** или **b** предварительно вводятся с клавиатуры. На печать выводятся только те значения **z**, величина которых превышает значение **a**. Цикл должен быть построен с помощью условного оператора **if**. Проверять возможность проведения вычислений (В частности деления на нуль при значении $x=0$). Построить и разместить в отчете блок-схему программы с оператором **if**.

5.6. 5.6 Цикл вычисления значений функции по варианту операторы (for, do, while)

Выполнить предыдущее задание вычисления значений функции с использованием операторов циклов for, do или while (см. свои варианты). Значения переменных **a** и **b** вычислить в программе. Результат распечатать в том же цикле. Построить и разместить в отчете блок-схему программы с оператором цикла.

6. 6. Контролируемые требования.

- Примеры раздела основные понятия данных МУ.
- Фрагмент программы для минимума из трех целых чисел
- Вычисление корней уравнения с использованием переключателя
- Цикл с помощью оператора if
- Табуляция функции с помощью оператора цикла
- Задания раздела дополнительных требований (раздел 8).
- Требования и рекомендации к оформлению отчета по ЛР.

7. 7. Варианты заданий для студентов СУЦ.

Варианты заданий приведены ниже. Номер варианта должен соответствовать номеру студента в групповом журнале. Значения **b** вычислить в программе до цикла, а значение **a** ввести перед циклом.

Варианты для циклических программ (номер должен соответствовать номеру группового журнала, уточните у старосты группы).

1. $z=\cos(bx)/x$, $x=0,1,\dots,10$. ОПЕРАТОР ЦИКЛА **for**.
2. $z=\sin(bx)/x$, $x=0.1, 0.2,\dots,1$. ОПЕРАТОР ЦИКЛА **while**.
3. $z=\text{tg}(bx)/x$, $x=0,1,\dots,10$. ОПЕРАТОР ЦИКЛА **for**.
4. $z=\cos(0.1bx)/x$, $x=0.1, 0.2,\dots,1$. ОПЕРАТОР ЦИКЛА **while**.
5. $z=\cos(bx/10)/x$, $x=0.1, 0.2,\dots,1$. ОПЕРАТОР ЦИКЛА **for**.
6. $z=\cos(bx*x)/x$, $x=0, 1,\dots,10$. ОПЕРАТОР ЦИКЛА **while**.
7. $z=\text{tg}(bx)/x$, $x=-0.1, -0.2,\dots,-1$. ОПЕРАТОР ЦИКЛА **for**.
8. $z=\cos(bx)/x$, $x=0, -1,\dots,-10$. ОПЕРАТОР ЦИКЛА **do**.
9. $z=\sin(0.1bx)/x$, $x=0, 1,\dots,10$. ОПЕРАТОР ЦИКЛА **do**.
10. $z=\arcsin(bx)/x$, $x=-0.1, -0.2,\dots,-1$. ОПЕРАТОР ЦИКЛА **for**.

Примечание 1.

Если при компиляции выдается сообщение вида:

error C2668: cos: неоднозначный вызов перегруженной функции

Для обращения к функции cos:

`z=cos(2*a); // Ошибка a - int`

То необходимо выполнить согласование типов (float/double) с использованием **cast** операторов (**double**) для преобразований или трансформирования констант в заданном выражении в действительную константу (точка в числе – **2.0**). Например:

`z=cos(2.0*a); // использование действительной константы`

`z=cos((double)2*a); // использование cast- преобразований`

8. 8. Дополнительные требования для сильных студентов СУЦ (д.т.).

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

8.1. 7.1 Определение максимального значения

Написать программу для определения максимального значения из трех целых переменных: А, В, С. Значения переменных вводятся с клавиатуры. При выводе учесть факт возможного равенства значений переменных, делая при этом дополнительные проверки. (всего должно 8 различных вариантов для выхода при сравнении - **2³**) В выводе может быть даже такой вариант:

```
printf("Максимально (A = B = C) - %d\n" , B);
```

8.2. 7.2 Вычисления с запоминанием значений

Написать программу для вычисления значений функции по формуле контрольного задания и запомнить результаты вычисления в действительном (типа double) массиве **MasZ (double)**, а соответствующие вычисленные в программе аргументы в массиве **MasX (double)**. Размер массива определен заранее (но не более 10 элементов). Распечатать оба массива после завершения вычислений столбиком. Значения **b** для формулы перед циклом ввести с клавиатуры. Использовать оператор цикла по варианту (**for** или **while**). Быть готовым переделать свою программу для оператора цикла **do**.

8.3. 7.3 Вычисления с запоминанием значений и вводом аргументов

Написать программу для вычисления значений функции по формуле контрольного задания и запомнить результаты вычисления в массиве действительном **MasZ** (типа double). Максимальный размер массива определен заранее (≤ 10). Значения нужно **b** вычислить в программе. Исходные данные (аргументы **x**) предварительно и последовательно ввести с клавиатуры в отдельный массив **MasX** (типа float). Распечатать весь вычисленный массив **MasZ** и соответствующие аргументы из массива **MasX** после завершения вычислений в виде таблицы. Использовать оператор цикла **while**. Контролировать превышение размера массива при вводе.

8.4. 7.4 Вычисления минимума в массиве

Написать программу для вычисления минимума **MinM** в целочисленном (int) массиве **MasM**. Размерность массива задана (10). Значения элементов массива заданы при его описании (инициализация массива). Значение минимума **MinM** и его номер в массиве вывести на экран. Использовать оператор цикла **for**.

Примечание: Выполнение программы с дополнительными требованиями фиксируется в журнале учета ЛР и в отчете по ЛР и окончательно учитывается при подведении результатов работы в семестре и на экзамене (автоматы).

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. Для чего нужны проекты и в чем их преимущество (три)?
2. Какие изменения последовательности выполнения программы вы знаете?
3. Почему нужны ветвления в программе?
4. Какие операторы управления программой вы знаете?
5. Что такое цикл?
6. Какие составляющие циклов вы знаете?
7. Какие операторы управления программой вы знаете?
8. Поясните работу оператора goto.
9. Поясните работу оператора if.
10. Поясните работу оператора switch.
11. Поясните работу оператора for.
12. Поясните работу оператора do.
13. Поясните работу оператора while.
14. Как построить цикл с помощью условного оператора?
15. Могут ли быть вложенными условные операторы?
16. Могут ли быть вложенными циклы?
17. В чем преимущество возможности ветвления в программах?
18. Для чего используется оператор break?
19. Для чего используется оператор continue?
20. 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. К Язык программирования C, 2/3-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!). Или online в 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. Хананова, 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. 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.
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.

12.11. Болванки для ЛР, демонстрации ЛР и ДЗ.

12.1.11.11. 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
// 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 из МУ
system(" PAUSE");
}
```

12.2.11.11. Болванка для демонстрации ЛР или работы на семинаре

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 консольное окно и выполнить пошагово.

```
#define _USE_MATH_DEFINES
// 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 из МУ
////////////////////
// 1-й СПОСОБ для демонстрации и занятий
// для использования может быть выбран один или второй способ (взаимоисключающие!)

goto METBEGIN;

// НАЧАЛО

// 1
METBEGIN:; // Метку перетащить к началу составного оператора проверяемого пункта или за-
дания
{
    // п 5.6
    //
goto METEND;} ;
// 2

{ // п 5.7
goto METEND;} ;
// ...

{
    //
goto METEND;} ;
// ...
// КОНЕЦ
METEND:; // Метка конца проверки
// 2-й СПОСОБ для демонстрации и занятий

char SW;
// Вывод меню
printf("1. П. 5.6\n");
printf("2. П. 5.7\n");
// ...
// выбрать нужный пункт меню
printf("Выберете пункт меню:\n");
```

```
SW = getchar(); // Ввод нажатой клавиши
// Переключатель
```

```
switch ( SW)
{
    case '1':
    {
        printf("Выбор 1!\n");
        // проверяемая программа
        // п.5.6
        break;
    }
    case '2':
    {
        printf("Выбор 2!\n");
        //проверяемая программа
        //п.5.7
        break;
    }
    case '3':
    {
        printf("Выбор 3!\n");
        break;
        break;
    }

    default:
    {
        printf("Выбор по умолчанию!!\n");
    }

};
```

```
system(" PAUSE");
}
```

12.3.11.3. Болванка с меню, выходом и возвратом в меню

В данном разделе приводиться текст “болванки с меню, возможностью выхода из программы и возврата в меню”

```
#define _USE_MATH_DEFINES
// 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 из МУ
    //////////////////////////////////
    // 1-й СПОСОБ для демонстрации и занятий
    // для использования может быть выбран один или второй способ (взаимоисключающие!)
    goto METEND; // При использовании 2-го способа !!!
    goto METBEGIN;

    // НАЧАЛО
```

```

// 1
METBEGIN;; // Метку перетащить к началу составного оператора проверяемого пункта или задания
{
    // п 5.6
    //
goto METEND;};
// 2

{ // п 5.7
goto METEND;};
// ...

{
    //
goto METEND;};
// ...
// КОНЕЦ
METEND;; // Метка конца проверки
// 2-й СПОСОБ для демонстрации и занятий

char SW;

MENU;;
// Вывод меню
system (" CLS");
MENU2;;
printf("1. П. 5.6...\n");
printf("2. П. 5.7...\n");
printf("3. П. 5.8...\n");
// ....
printf("е. Выход ...\n");
printf("0. Меню ...\n");

// ...
// выбрать нужный пункт меню
printf("Выберете пункт меню:\n");
SW = getchar(); // Ввод нажатой клавиши
// Переключатель

switch ( SW)
{
    case '1':
    {
printf("Выбор 1!\n");
// проверяемая программа - после этого пункта завершение проверки и программы !!
// п.5.6 ...
break;
    }

    case '2':
    {
printf("Выбор 2!\n");
//проверяемая программа - после этого пункта выход в меню
//п.5.7 ...
// system (" CLS");
SW = getchar(); // Сброс ENTER
goto MENU;
break;

```

```

    }
    case '3':
    {
printf("Выбор 3!\n");
//проверяемая программа - пункта выход в меню с сохранением результата
//п.5.8 ...

SW = getchar(); // Сброс ENTER
goto MENU2;
break;

    }
    case '0':
    {
printf("menu!\n");
// После этого пункта очистка и возврат в меню
SW = getchar(); // Сброс ENTER
goto MENU;
break;
break;

    }
    case 'e':
    {
printf("Выбор е !\n");
// После этого пункта выход из программы
exit (0);
    }
default:
    {
printf("Выбор по умолчанию!!\n");
    }

};

system(" PAUSE");
}

```

12.4.3.19. Болванка с меню, выходом и возвратом в меню для ДЗ

```

#define _USE_MATH_DEFINES
// Подключение библиотеки математических функций
#include <math.h>
#include <stdio.h>
#include <process.h>
#include <process.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
#include <time.h>
#include <io.h>
/// ВРЕМЕННО ЗАКОМЕНТИРОВАНО #include "header.h"
extern int Counter;
void main(void)
{
// // п.
system(" chcp 1251 > nul");
// Здесь расположить тексты примеров из МУ

```

```

////////////////////
// 2-й СПОСОБ для демонстрации и занятий

char SW;

MENU::
// Вывод меню
system (" CLS");
MENU2::
printf("1. П. ТЗ 5.1.1 ...\\n");
printf("2. П. ТЗ 5.1.3 ...\\n");
printf("3. П. ТЗ 5.1.3 ...\\n");
printf("4....\\n");
printf("* П. ТЗ 5.X.X ...\\n");
// ....
printf("в/у/е. Выход ...\\n");
printf("0. Меню ...\\n");

// ...
// выбрать нужный пункт меню
printf("Выберете пункт меню:\\n");
SW = getchar(); // Ввод нажатой клавиши
// Переключатель

switch ( SW)
{
    case '1':
    {
        printf("Выбор 1!\\n");
        // проверяемая программа - после этого пункта завершение проверки и программы !!
        // п.5.6 ...
        break;
    }

    case '2':
    {
        printf("Выбор 2!\\n");
        //проверяемая программа - после этого пункта выход в меню
        //п.5.7 ...
        // system (" CLS");
        SW = getchar(); // Сброс ENTER
        goto MENU;
        break;
    }

    case '3':
    {
        printf("Выбор 3!\\n");
        //проверяемая программа - пункта выход в меню с сохранением результата
        //п.5.8 ...

        SW = getchar(); // Сброс ENTER
        goto MENU2;
        break;
    }
}

////////////////////
/// Шаблоны для case
/*
///// Шаблон без очистки результата !!!!! убрать комментарии и заменить *

```

```

        case '*':
        {
printf("Выбор *!\n");
//проверяемая программа - пункта выход в меню без сохранения результата
//п.ТЗ ...

SW = getchar(); // Сброс ENTER
goto MENU;
break;
        }

        /*
        /*
        /*
        /* Шаблон с очисткой результата !!!!! убрать комментарии и заменить *
        case '*':
        {
printf("Выбор *!\n");
//проверяемая программа - пункта выход в меню с сохранением результата
//п.ТЗ ...

SW = getchar(); // Сброс ENTER
goto MENU2;
break;
        }

        /*
        /*
        /*
        /*убрать комментарии

//////////
case '0':
{
printf("menu!\n");
// После этого пункта очистка и возврат в меню
SW = getchar(); // Сброс ENTER
goto MENU;
break;
break;
}
case 'y':
case 'e':
case 'в':
{
printf("Выбор в !\n");
// После этого пункта выход из программы
exit (0);
}
default:
{
printf("Выбор по умолчанию!!\n");
}

};

system(" PAUSE");
}

```


При включении в проект своего заголовочного файла "header.h" нужно его создать, подключить в проект и убрать слешы (//) в его временном отключении в начале этого фрагмента.