Кыргызко-Российский Славянский университет

Кафедра информатики и вычислительной техники

**Сычев В.Н., Сычева Н.А.**

**Программирование на языке Си**

**Лабораторный практикум для студентов специальности**

**«Программная инженерия»**

Бишкек 2014

Сычев В.Н, доцент кафедры ИВТ, ЕТФ, КРСУ, канд.физ.мат.наук.

Сычева Н.А., доцент кафедры ИВТ, ЕТФ, КРСУ, канд.физ.мат.наук.

Цель практикума закрепить знания, полученные при изучении теоретической части курса и получить практические навыки разработки проектирования и реализации алгоритмов решения практических задач на языке С++.Пособие включает в себя 18 практических занятий. Каждое занятие содержит тему, краткую информацию по теме и заданияпо рассматриваемое теме. Практикум предназначен для студентов очной и заочной формы обучения по специальности «Программная инженерия».

Под редакцией зав.кафедрой ИВТ, ЕТФ, КРСУ, профессора, Десяткова Г.А.

Оглавление

[Вопросы по курсу Си. 4](#_Toc380551008)

[Введение 5](#_Toc380551009)

[Практическое занятие № 1 . 6](#_Toc380551010)

[Практическое занятие № 2 7](#_Toc380551011)

[Практическое занятие № 3 8](#_Toc380551012)

[Практическое занятие № 4 9](#_Toc380551013)

[Практическое занятие № 5 10](#_Toc380551014)

[Практическое занятие № 6 11](#_Toc380551015)

[Практическое занятие № 7 11](#_Toc380551016)

[Практическое занятие № 8 12](#_Toc380551017)

[Практическое занятие № 9 13](#_Toc380551018)

[Практическое занятие № 10 15](#_Toc380551019)

[Практическое занятие № 11 16](#_Toc380551020)

[Практическое занятие № 12 17](#_Toc380551021)

[Практическое занятие № 13 19](#_Toc380551022)

[Практическое занятие № 14 21](#_Toc380551023)

[Практическое занятие № 15 21](#_Toc380551024)

[Практическое занятие № 16 22](#_Toc380551025)

[Практическое занятие № 17 23](#_Toc380551026)

[Практическое занятие №18 23](#_Toc380551027)

# Вопросы по курсу Си.

1. Трансляция, компоновка, библиотеки. Препроцессор. Директивы препроцессора
2. Спецификаторы ***const*** и ***volatile***.
3. Базовые типы данных. Константные величины. Целочисленные константы и константы с плавающей точкой. Символьные константы. Строковые константы.
4. Переменные. Определения и объявления. Определение переменной. Инициализация. Спецификатор const. Тип перечисление.
5. Арифметические операции. Операции отношения и логические операции. Операции присваивания. Операции инкремента и декремента (увеличения и уменьшения). Операция sizeof.
6. Условная операция. Операция запятая. Побитовые (поразрядные) операции.
7. Преобразования типа. Неявные преобразования типа. Явные преобразования типа.
8. Операторы. Составные операторы и блоки. Оператор ***if-else***. Оператор ***switch*** (переключатель). Операторы перехода ***break***, ***continue***, ***goto***.
9. Операторы. Операторы цикла. Оператор ***while***. Оператор ***do-while***. Оператор ***for***. Операторы перехода ***break***,***continue***,***goto***.
10. Массивы. Определение и использование массивов. Многомерные массивы.
11. Указатели. Определение и использование указателей. Константные указатели. Указатели и массивы. Адресная арифметика. Динамическое распределение памяти. Символьные указатели и строки символов. Массивы указателей. Указатели на указатели.
12. Ссылки. Описания ***typedef***. Абстрактные имена типов.
13. Структуры и объединения. Определение и использование структур. Структуры со ссылками на себя. Объединения (смеси). Битовые поля.
14. Функции. Определение функции. Вызов функции. Прототипы функции. Строгая проверка типа. Возвращаемое значение. Список параметров функции. Произвольное число параметров. Параметры по умолчанию. Способы передачи параметров. Передача по значению. Параметры - указатели. Способы передачи параметров. Параметры - массивы. Многомерные массивы как параметры. Cпособы возврата значения. Функции - подстановки. Указатели на функции.
15. Правила чтения сложных описаний.
16. Области видимости. Виды областей видимости. Файловая область видимости. Операция глобальной области видимости. Объявления внешних переменных. Статические глобальные переменные и функции. Локальная область видимости. Автоматические локальные переменные. Пространство имен.
17. Структура программы и использование заголовочных файлов. Видимость объявлений типов.
18. Правила использования заголовочных файлов. Подходы к разработке структуры программы.
19. Перегрузка (совместное использование) имен функций. Объявление перегруженной функции. Сопоставление параметров. Точное сопоставление. Расширение типа. Стандартные преобразования. Сопоставление функций с несколькими параметрами. Сопоставление функций с параметрами по умолчанию. Области видимости перегруженных функций. Указатели на перегруженную функцию.

# Введение

Си (англ. C) — язык программирования, разработанный в 1969—1973 годах сотрудниками BellLabs Кеном Томпсоном и ДеннисомРитчи как развитие языка Би. Первоначально был разработан для реализации операционной системы UNIX, но, впоследствии, был перенесён на множество других платформ. Благодаря близости по скорости выполнения программ, написанных на Си, к языку ассемблера, этот язык получил широкое применение при создании системного программного обеспечения и прикладного программного обеспечения для решения широкого круга задач. Язык программирования Си оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования как C++, C# и Java.

Язык программирования Си отличается минимализмом. Авторы языка хотели, чтобы программы на нём легко компилировались с помощью однопроходного компилятора, чтобы каждой элементарной составляющей программы после компиляции соответствовало весьма небольшое число машинных команд, а использование базовых элементов языка не задействовало библиотеку времени выполнения. Однопроходный компилятор компилирует программу, не возвращаясь назад, к уже обработанному тексту. Поэтому использованию функции и переменных должно предшествовать их объявление. Код на Си можно легко писать на низком уровне абстракции, почти как на ассемблере. Иногда Си называют «универсальным ассемблером» или «ассемблером высокого уровня», что отражает различие языков ассемблера для разных платформ и единство стандарта Си, код которого может быть скомпилирован без изменений практически на любой модели компьютера. Си часто называют языком среднего уровня или даже низкого уровня, учитывая то, как близко он работает к реальным устройствам. Однако, в строгой классификации, он является языком высокого уровня.

Компиляторы Си разрабатываются сравнительно легко благодаря простоте языка и малому размеру стандартной библиотеки. Поэтому данный язык доступен на самых различных платформах (возможно, круг этих платформ шире, чем у любого другого существующего языка). К тому же, несмотря на свою низкоуровневую природу, язык позволяет создавать переносимые программы и поддерживает в этом программиста. Программы, соответствующие стандарту языка, могут компилироваться на самых различных компьютерах.

Си (как и ОС UNIX, с которой он долгое время был связан) создавался программистами и для программистов, круг которых был бы ненамного шире круга разработчиков языка. Несмотря на это, область использования языка значительно шире задач системного программирования.

Си создавался с одной важной целью: сделать более простым написание больших программ с минимумом ошибок по правилам процедурного программирования, не добавляя на итоговый код программ лишних накладных расходов для компилятора, как это всегда делают языки очень высокого уровня, такие как Бейсик. С этой стороны Си предлагает следующие важные особенности:

* простую языковую базу, из которой вынесены в библиотеки многие существенные возможности, вроде математических функций или функций управления файлами;
* ориентацию на процедурное программирование, обеспечивающую удобство применения структурного стиля программирования;
* систему типов, предохраняющую от бессмысленных операций;
* использование препроцессора для, например, определения макросов и включения файлов с исходным кодом;
* непосредственный доступ к памяти компьютера через использование указателей;
* минимальное число ключевых слов;
* передачу параметров в функцию по значению, а не по ссылке (при этом передача по ссылке эмулируется с помощью указателей);
* указатели на функции и статические переменные;
* области действия имён;
* структуры и объединения — определяемые пользователем собирательные типы данных, которыми можно манипулировать как одним целым;

Одним из последствий высокой эффективности и переносимости Си стало то, что многие компиляторы, интерпретаторы и библиотеки других языков высокого уровня часто написаны на языке Си.

# Практическое занятие № 1 .

***Тема:*** *Базовые типы данных. Константные величины. Целочисленные константы и константы с плавающей точкой. Символьные константы. Строковые константы.*

***Объявление переменных***

* Каждая переменная программы должна быть объявлена.
* Объявление переменных обычно помещают в начале функции, сразу за заголовком. Следует обратить внимание, что хотя язык С++ допускает объявление переменных практически в любом месте программы, объявлять переменные всё-таки лучше в начале функции, снабжая инструкцию объявления кратким комментарием о назначении переменной.
* Инструкция объявления переменной выглядит так:*ТипИмяПеременной*;
* Инструкцию объявления переменной можно использовать для инициализации переменной. В этом случае объявление переменной выглядит так:

*Тип ИмяПеременной=НачальноеЗначение;*

* В имени переменной можно использовать буквы латинского алфавита и цифры (первым символом должна быть буква).
* Компилятор С++ различает прописные и строчные буквы, поэтому например, имена Summa и summa обозначают разные переменные.
* Основными числовыми типами языка С++ являются целый и с плавающей точкой.
* После инструкции объявления переменной рекомендуется указать назначение переменной.

***Вывод***

* Функция *printf* обеспечивает вывод на экран монитора сообщений и значений переменных.
* Первым параметром функции *printf* является строка вывода, определяющая выводимый текст и формат вывода значений переменных, имена которых указаны в качестве остальных параметров.
* Формат вывода значений переменных задается при помощи спецификатора преобразования- последовательности символов, начинающихся с символа %.
* При выводе числовых значений наиболее часто используются следующие спецификаторы: %d для вывода целых чисел со знаком, %u- для вывода целых беззнаковых целых, %f – для вывода дробных, в виде числа с плавающей точкой, %n.mf для вывода дробных в формате с фиксированной точкой, где n – количество символов всего и m в дробной части.
* Некоторые символы могут быть помещены в строку вывода только как последовательность других, обычных символов: \n – новая строка, \t – табуляция,\” – двойная кавычка,\\ - символ \.
* Наряду с функцией printf, для вывода на экран сообщений можно использовать функцию puts, которая после вывода текста автоматически переводит курсор на начало следующей строки.
* Чтобы сразу после окончания работы программы окно, в котором программа работала, не было автоматически перекрыто другим окном, например окном редактора текста среды разработки в конце программы нужно вставить следующие инструкции:

*printf(“для завершения работы нажмите клавишу <Enter>”);*

*getch();*

В языке существуют следующие виды констант:

* целые(int), например 125;
* вещественные с двойной точностью (double), например 123.456, 5.61е-4. Они могут снабжаться суффиксом F (или f) для типа float, например 123.456F, 5.61e-4f;
* короткие целые (short), в конце записи которых добавляется буква (суффикс) H (или h), например 275h, 344H;
* длинные целые (long), в конце записи которых добавляется буква (суффикс) L (или l), например 361327L;
* беззнаковые (unsignedint), в конца записи которых добавляется буква U (или u), например 62125U;
* восьмеричные (int), в которых перед первой значащей цифрой записывается нуль (0), например 071;
* шестнадцатеричные (int), в которых перед первой значащей цифрой записывается пара символов нуль-икс (0x), например 0x5F;
* символьные (char) - единственный символ, заключенный в одинарные кавычки, например 'О', '2', '.' и т.п. Символы, не имеющие графического представления, можно записывать, используя специальные комбинации, например \n (для новой строки), \0 (код которого 00000000). Эти комбинации выглядят как два символа, хотя фактически это один символ. Так же (можно представить любой двоичный образ одного байта: '\NNN', где NNN - от одной до трех восьмеричных цифр. Допускается и шестнадцатеричное задание кодов символов, которое представляется в виде: '\х2В', '\хЗ6' и т.п.;
* строковые (*constchar \**)- последовательность из нуля символов и более, заключенная в двойные кавычки, например: "Это строковая константа". Кавычки не входят в строку, а лишь ограничивают ее. Строка представляет собой массив из перечисленных элементов, в конце которого помещается байт с символом '\0'. Таким образом, число байтов, необходимых для хранения строки, на единицу превышает число символов между двойными кавычками;
* константное выражение, состоящее из одних констант, которое вычисляется во время трансляции (например: а=60+301);
* типа *longdouble*, в конце записи которых добавляется буква L (или l), например: 1234567.89L.

***Задание:***

1. Написать программу, которая объявляет, инициализирует и выводит на экран базовые типы данных: *char, int, long, float, double, longdouble*. Использовать функцию *printf* для вывода значений переменных на экран.
2. Написать программу, которая выводит на экран следующую таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип | Байт | Бит | Мин. | Мак. |
| char |  |  |  |  |
| int |  |  |  |  |
| long |  |  |  |  |
| float |  |  |  |  |
| double |  |  |  |  |
| long double |  |  |  |  |

Для вывода количества байт использовать операцию***sizeof*** и для вычисления *min* и *max* целочисленных типов использовать математическую функцию *pow*.

***Задание на дом***: лабораторная работа №1

# Практическое занятие № 2

***Тема:*** *Арифметические операции. Операции отношения и логические операции. Операции присваивания. Операции инкремента и декремента (увеличения и уменьшения).*

Операции ***присваивания*(=,+=, - =, \*=, /=, %=, <<=,>>=, &=, |=, ^=,++,--).**

* Операция присваивания предназначена для изменения значений переменных, в том числе и для вычислений «по формуле».
* присваивание является не оператором, а операцией, которая может находиться на любом месте в выражении.
* В С++ операция присваивания, выполняющая некоторое действие, может быть описана несколькими способами, например, вместо x=x+dx можно записать x+=dx, а вместо i=i+1 воспользоваться оператором инкремента и записать i++.
* Значение выражения в левой части инструкции присваивания зависит от типа операндов и операции, выполняемой над операндами. Целочисленное сложение и вычитание выполняется без учета переполнения. Например, если переменная n объявлена как intв 16-тиразрядной ОС, имеет значение 32767, то в результате выполнения инструкции n=n+1, значение переменной n будет равно -32768.
* Результатом выполнения операции деления над целыми операндами является целое число, которое получается отбрасыванием дробной части результата деления.

***Ввод***

* Для ввода исходных данных с клавиатуры используется функция scanf.
* Первым параметром функции scanfявляется управляющая строка, остальные параметры- адреса переменных, значения которых должны быть введены.
* Управляющая строка представляет собой заключенный в двойные кавычки список спецификаторов: %d для ввода целыхчисел со знаком, %u- для ввода целых беззнаковых целых, %f – для ввода вещественных чисел, %c – для ввода символа, %s для ввода строки.
* Использование имени переменной, а не ее адреса в качестве параметров функции scanfявляется типичной ошибкой начинающих программистов. Компилятор эту ошибку не обнаруживает.

***Задание:***

1. Напишите программу расчета y и z по формулам:



Определить разность между значениями y и z. Обеспечьте ввод α склавиатуры.

***Задание на дом*:** лабораторная работа №2

# Практическое занятие № 3

***Тема:****Побитовые(*поразрядные*)операции.*

* Поразрядные операторы позволяют работать с числами на уровне отдельных битов.
* Операнды побитовых операций должны быть целочисленного типа. Хотя они могут быть знаковыми и беззнаковыми, предпочтительнее беззнаковый тип с точки зрения переносимости, т.к. обработка побитовой операцией "знакового разряда" числа может отличаться в различных реализациях языка.
* Для получения результата поразрядные операторы выполняют операции булевой алгебрыс соответствующими битами аргументов.
* **~** : поразрядное отрицание, или побитовое НЕ,или дополнение до единицы. Эта унарная операция изменяет каждую 1 на 0, а 0 на 1. Поэтому ~(10011010) == (01100101).
* **&**: Поразрядное И. Сравнивает последовательно разряд за разрядом два операнда. Для каждого разряда результат равен 1, если только оба соответствующих разряда операндов равны 1. Так: (10010011) & (00111101) == (00010001).
* **|** : Поразрядное ИЛИ. Сравнивает последовательно разряд за разрядом два операнда. Для каждого разряда результат равен 1, если любой из соответствующих разрядов операндов равен 1. Так: (10010011) | (00111101) == (10111111).
* **^**: Поразрядное исключающее ИЛИ. Сравнивает последовательно разряд за разрядом два операнда. Для каждого разряда результат равен 1, если один нз двух (но не оба) соответствующих разрядов операндов равен 1. Поэтому (10010011) ^ (00111101) == (10101110).
* **<<** : Сдвиг влево. Сдвигает разряды левого операнда влево на число позиций, указанное правым операндом. Освобождающиеся позиции заполняются нулями, а разряды, сдвигаемые за левый предел левого операнда, теряются. Поэтому (10001010) << 2 == 00101000, где каждый разряд сдвинулся на две позиции влево.
* **>>** : Сдвиг вправо. Сдвигает разряды левого операнда вправо на число позиций, указанное правым операндом. Разряды, сдвигаемые за правый предел левого операнда, теряются. Для чисел типа unsigned позиции, освобождающиеся слева, заполняются нулями. Для чисел со знаком результат зависит от типа ЭВМ. Освобождающиеся позиции могут заполняться нулями или значением знакового разряда (самого левого). Для значений без знака имеем(10001010) >> 2 == (00100010), где каждый разряд переместился на две позиции вправо.
* Поразрядные операторы могут объединяться со знаком = для совмещения операции с присваиванием. Операции &=, |=, ^= ,>>=, <<= являются допустимыми.

***Задание:***

1. Написать программу удвоения любого целого числа с помощью операции сдвига. Число вводить с клавиатуры.
2. Используя только побитовые операторы изменить знак числа, который вводит пользователь.
3. Используя только побитовые операторы представить число, которое вводит пользователь в двоичном, восьмеричном и шестнадцатеричном виде.
4. Написать программу сложения двух целых (int) чисел побитно используя поразрядные операции. Результат операции сложения вывести на экран. Подсчитать и вывести количество нулей и единиц в слагаемых и в сумме.

***Задание на дом***: лабораторная работа №3

# Практическое занятие № 4

***Тема:*** *Условный оператор* ***if*** *и оператор* ***switch*** *в языке Си.*

***Операторif***

***if*** (выражение)оператор;

***else***оператор;

* Если выражение истинно, то выполняется первый оператор, а если ложно, то выполняется оператор, следующий за ключевым словом ***else***. ***else***-часть может отсутствовать. Операторы могут быть простыми или составными.
* Если при соблюдении или несоблюдении условия надо выполнить несколько операторов программы, то эти операторы следует объединить в группу – заключить в фигурные скобки.
* При помощи вложенных одна в другую несколько инструкций ***if***можно реализовать множественный выбор.

***Операторswitch***

***switch(целоe выражение)***

{

***case*** константа1 : операторы; (необязательные)

***case*** константа2 : операторы; (необязательные)

***default*** (необязательные) : операторы; (необязательные)

***}***

* Оператор***switch*** предназначена для выбора одного из нескольких возможных направлений дальнейшего хода программы.
* Выбор последовательности операторов осуществляется в зависимости от равенства значения значение выражения константе, указанной в слове ***case***.
* Если значение выражения не равно ни одной из констант, записанных после слова case, то выполняются инструкции, расположенные после слова ***default***.
* Метки, имеющиеся в операторе ***switch***, должны быть константами или константными выражениями (выражения, операнды которого константы) целого типа (включая тип ***char***). Запрещается использовать в качестве метки переменную.
* Значением выражения в скобках должна быть величина целого типа (опять же, включая тип ***char***).
* Выполнение оператора ***break*** приводит к тому, что в программе происходит выход из оператора ***switch*** и осуществляется переход к следующему за ним оператору. При отсутствии оператора ***break*** будут выполнены все операторы, начиная с помеченного данной меткой и завершая оператором ***switch***.

***Задание:***

1. Написать программу, которая переводит время из часов, минут и секунд в секунды. Программа должна проверять правильность введенных пользователем данных и в случае, если данные неверные выводить соответствующее сообщение.
2. Написать программу, котораязапрашивает у пользователя номер дня недели, затем выводит название дня недели или сообщение об ошибке, если введены неверные данные.

***Задание на дом*:** лабораторная работа №4 (желательно графическое представление решения в тестовом режиме**)**

# Практическое занятие № 5

***Тема:****Операторы цикла в языке Си.*

Циклы организуются, чтобы выполнить некоторый оператор или группу операторов определенное число раз. В языке Си три оператора цикла: ***for***, ***while*** и ***do - while.***

***for****(инициализация; проверка условия; коррекция) оператор;*

*Инициализирующее выражение* вычисляется только один раз до начала выполнения какого-нибудь из операторов цикла. Если *проверяемое* выражение оказывается истинным (или не равным нулю), тело цикла выполняется один раз. Затем вычисляется величина *корректируемого* выражения, и значение *проверяемого* выражения определяется вновь. Оператор ***for*** - это цикл с предусловием: решение, выполнить в очередной раз тело цикла или нет, принимается до начала его прохождения. Поэтому может случиться так, что тело цикла не будет выполнено ни разу. *Оператор*, образующий тело цикла, может быть как простым, так и составным.Любое из трех выражений в цикле ***for*** может отсутствовать, однако точка с запятой должна оставаться.

Тело цикла выполняется до тех пор, пока проверяемое условие не станет ложным или равным нулю

Примеры:

*for (i = 1; i< 10; i++) { ... }*

*for (сh = 'a'; ch != 'p';) scanf ("%c", &ch);*

*/\* Цикл будет выполняться до тех пор, пока с клавиатуры*

*не будет введен символ 'p' \*/*

***while****(выражение) тело\_цикла*

Выражение в скобках может принимать ненулевое (истинное) или нулевое (ложное) значение. Если оно истинно, то выполняется тело цикла и выражение вычисляется снова. Если выражение ложно, то цикл while заканчивается.

***do****{тело\_цикла}****while****(выражение);*

Основным отличием между циклами ***while*** и ***do - while*** является то, что тело в цикле ***do - while***выполняется по крайней мере один раз. Тело цикла будет выполняться до тех пор, пока выражение в скобках не примет ложное значение. Если оно ложно при входе в цикл, то его тело выполняется ровно один раз.

Допускается вложенность одних циклов в другие, т.е. в теле любого цикла могут появляться операторы ***for, while и do - while.***

В теле цикла могут использоваться новые операторы ***break*** и ***continue***. Оператор ***break*** обеспечивает немедленный выход из цикла,

***Задание*:**

1. Написать программу, которая выводиттаблицу степеней двойки от нулевой до десятой.
2. Написать программу вычисления числа **π** с заданной точностью (например: 0.00001 или 1e-5) вводимой с клавиатуры, которое определяется суммой ряда вида (1-1/2+1/3-1/4+..1/(2\*n-1))\*4. Подсчитать количество членов ряда, которое дает заданную точность вычисления.

***Задание на дом*:** лабораторная работа №5

# Практическое занятие № 6

***Тема:*** *Массивы. Определение и использование массивов. Одномерные массивы.*

**Массивы**

* Массив – это структура данных, представляющая собой набор, совокупность элементов одного типа.
* В инструкции объявления массива указывается количество элементов массива.
* Элементы массива нумеруются с нуля.
* Доступ к элементу массива осуществляется путем указания индекса (номера) элемента. В качестве индекса можно использовать выражение целого типа – константу или переменную. Индекс может меняться от 0 до n-1, где n- количество элементов массива.
* Доступ к элементу массива можно осуществить при помощи указателя.
* В инструкции объявления массива удобно использовать константу.
* Для ввода и вывода и обработки массива удобно использовать инструкции циклов (***for***, ***while***и ***do*** - ***while***).
* Типичной ошибкой при использовании массивов является обращение к несуществующему элементу, т.е. выход индекса за допустимое значение.

***Задание*:**

1. Написать программу вывода на экран названия дня недели по дате (год, месяц, день), введенной с клавиатуры, если известно, что 1 января 1800 года была среда. Год считается високосным, если его значение делится на 4, но не делится на 100 или делится и на 400. Условие високосности: if (year%4==0 && (year%100!=0 !! year%400==0))

***Задание на дом*:** лабораторная работа №6

# Практическое занятие № 7

***Тема:*** *Многомерные массивы.*

* Массивы можно использовать для представления таблиц значений, содержащих информацию, расположенную в виде строк и столбцов. Для идентификации отдельного элемента таблицы указываются два индекса: первый (по соглашению) определяет строку, в которой находится элемент, а второй (по соглашению) определяет столбец, содержащий этот элемент. Таблицы или массивы, требующие для идентификации отдельного элемента указания двух индексов, называются двумерными массивами.Двумерный массив представляется как одномерный, элементами которого так же являются массивы. По аналогии можно установить и большее число измерений.
* Когда мы передаем одномерный массив в качестве аргумента функции, в списке параметров функции скобки массива могут быть пусты. Размерность первого индекса многомерного массива можно не указывать, но размерности всех последующих индексов указывать необходимо. Компилятор использует эти размерности для того, чтобы определить расположение в памяти элементов многомерного массива.
* Чтобы передать функции одну строку двумерного массива, которая принимается как одномерный массив, просто передается имя массива и вслед за ним первый индекс.

***Задание:***

1. Написать программу, которая заполняет квадратную матрицу nxn (n – произвольное число от 3 до 9) случайными значениями и выводит на экран значения элементов матрицы. Вывести номер столбца, в котором находится минимальный элемент, номер строки, в которой находится максимальный элемент и сумму элементов матрицы, находящихся по главной и второстепенной диагонали.

***Задание на дом:* лабораторная работа №7**

# Практическое занятие № 8

***Тема:****Указатели. Определение и использование указателей. Константные указатели. Указатели и массивы. Адресная арифметика. Динамическое распределение памяти.*

* Указатели — это переменные, которые содержат в качестве своих значений адреса других переменных.
* Указатели должны быть объявлены до того, как они могут быть использованы.
* Объявление *int \*ptr;* объявляет ptr указателем на объект типа int и читается как «ptr - это указатель на int». Символ \* в объявлении указывает, что переменная является указателем.
* Для присваивания указателю начального значения можно использовать три значения: 0, NULL или адрес объекта того же типа. Присваивание указателю в качестве начального значения 0 или NULL эквивалентно.
* Единственным целым, которое может быть присвоено указателю, является 0.
* Операция адресации & возвращает адрес своего операнда.
* Операнд операции адресации должен быть переменной; операция адресации не применима к константам, выражениям, которые не возвращают ссылку.
* Операция \*, называемая операцией косвенной адресации или операцией разыменования, возвращает значение объекта, на который указывает ее операнд. Это называется разыменованием указателя.
* Если аргумент вызываемой функции должен ею изменяться, можно в качестве параметра передавать адрес аргумента. Вызванная функция затем может модифицировать значение аргумента в вызывающей функции, используя операцию разыменования \*.
* Функция, принимающая адрес в качестве аргумента, должна в качестве соответствующего формального параметра иметь указатель.
* Спецификатор const дает программисту возможность информировать компилятор о том, что значение данной переменной не должно модифицироваться.
* Массивы автоматически передаются ссылкой, использующей указатели, потому что значение имени массива является адресом массива.
* К числу арифметических операций, которые можно выполнять над указателями, относятся инкремент (++) указателя, декремент указателя (--), сложение (+ или += ) указателя и целого, вычитание (- или -= ) указателя и целого и вычитание одного указателя из другого, если указатели ссылаются на смежные участки памяти.
* При сложении или вычитании указателя и целого указатель увеличивается или уменьшается на величину, равную произведению этого целого на размер указанного объекта.
* Арифметические операции могут выполняться только с указателями, указывающими на смежные участки памяти, такие, как массив. Все элементы массива хранятся в памяти непосредственно друг за другом.
* При выполнении арифметических операций над указателем на массив символов результаты не отличаются от обычной арифметики, потому что каждый символ хранится в одном байте памяти.
* Указатели можно присваивать один другому, если оба указателя одного и того же типа. В противном случае нужно использовать приведение типов. Исключением из этого правила является указатель на *void*, который является обобщенным указателем, способным представлять указатели любого типа. Указателю на *void* можно присваивать все типы указателей без приведения типа. Однако указатель на *void* может быть присвоен указателю другого типа только явным приведением к типу соответствующего указателя.
* Указатель на *void* не может быть разыменован без явного приведения типа.
* Указатели можно сравнивать, используя операции проверки на равенство и отношения. Сравнение указателей имеет смысл только в случае, если они указывают на элементы одного и того же массива.
* Указатели можно индексировать точно так же, как имена массивов.
* Имя массива эквивалентно указателю на первый элемент массива.
* В записи указатель-смещение смещение — это то же самое, что индекс массива.
* Все выражения с индексами массива можно записать с помощью указателя и смещения, используя либо имя массива как указатель, либо отдельный указатель, указывающий на массив.
* Имя массива — это постоянный указатель, который всегда указывает на одну и ту же ячейку памяти. Но в отличие от обычных указателей имена массивов нельзя изменять.
* В языке программирования[Си](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))имеются следующие четыре[функции](http://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))для динамического распределения памяти, входящие в[стандартную библиотеку](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0):
* Эти функции имеют следующие описания:

*#include <stdlib.h>*

*void \*malloc (size\_t size); // от*[*англ.*](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)*memory allocation,выделениепамяти*

*void \*calloc (size\_tnum, size\_t size); // от*[*англ.*](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)*clearallocation,чистоевыделениепамяти*

*void \*realloc(void \*block, size\_t size); // от*[*англ.*](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)*reallocation,перераспределениепамяти*

*void free(void \*block); //(*[*англ.*](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)*free,освободить*

* В[C++](http://ru.wikipedia.org/wiki/C%2B%2B)имеются два оператора:[***new***](http://ru.wikipedia.org/wiki/New_(C%2B%2B))и[***delete***](http://ru.wikipedia.org/wiki/Delete_(C%2B%2B)).
* Выделение динамической памяти под двумерный массив целых чисел размера n на m:

*int \*\* a = new int\*[n];*

*for (inti = 0; i< n; ++i)*

*a[i] = newint[m];*

* Для освобождения памяти требуется выполнить:

*for (inti = 0; i< n; ++i)*

*delete [] a[i];*

*delete [] a;*

***Задание:***

1. Написать программу, которая выделяет память для хранения 100 целых элементов массива в динамической области, инициализирует значения элементов массива случайными числами и выводит на экран значения суммы элементов, среднее значение, минимальный и максимальный элемент массива.
2. Вывести на экран значения следующих элементов из предыдущей задачи: a[0], a[2], &a[5], \*a, \*(a+5), a+5, \*a+5, a. При выводе элементов сообщать информацию о том, что выводится на экран, например: «адрес второго элемента» или «значение третьего элемента».

***Задание на дом*:** лабораторная работа №8

# Практическое занятие № 9

***Тема:****Структуры и объединения. Определение и использование структур. Объединения (смеси). Битовые поля.*

* Структурой в языке C называется совокупность логически связанных переменных различных типов, сгруппированных под одним именем для удобства дальнейшей обработки..
* Ключевое слово ***struct*** начинает определение структуры. За словом ***struct*** следует имя типа, называемое тегом структуры (tag – ярлык, этикетка). Тело структуры заключается в фигурные скобки ({ и }). Каждое определение структуры должно заканчиваться точкой с запятой.
* Тэг (имя-этикетка) структуры может использоваться для объявления переменных данного типа структуры.
* Определения структуры не резервируют место в памяти; они создают новые типы данных, которые используются далее для объявления переменных.
* Доступ к полю структуры осуществляется с помощью операций доступа к элементу - операции точка (.) и операции стрелка (->).
* Операция точка обеспечивает доступ к элементу структуры посредством имени переменной объекта или ссылки на объект. Операция стрелка обеспечивает доступ к элементу структуры посредством указателя на объект.
* Объединение - это производный тип данных, элементы которого совместно используют одну и ту же область памяти. Элементы могут быть любого типа.
* Память, выделяемая для объединения, должна быть достаточной для того, чтобы в ней размещался самый большой элемент объединения. В большинстве случаев, объединения содержат два или больше типов данных. Только один элемент и, следовательно, только один тип данных может быть доступен в объединении в каждый данный момент времени.
* Объединения объявляются при помощи ключевого слова ***union*** в том же формате, что и структуры.
* Объединение может быть инициализировано только значением типа первого элемента объединения.

*union tag*

*{*

*inti;*

*double d;*

*} u={10};*

* Доступ к полю объединения осуществляется с помощью операций – · (точка) и -> (стрелка).

Битовые поля

Особую разновидность структур представляют собой битовые поля. Битовое поле - это последовательность соседних битов внутри одного, целого значения. Оно может иметь тип *signedint* или *unsignedint* и занимать от 1 до 16 битов. Поля размещаются в машинном слове в направлении от младших к старшим разрядам. Например, структура:

*struct prim {*

*int a:2;*

*unsigned b:3;*

*int c:5;*

*int d:1;*

*unsigned d:5;*

*} i, j;*

обеспечивает размещение данных в двух байтах (в одном слове). Если бы последнее поле было задано так: *unsigned* d:6, то оно размещалось бы не в первом слове, а в разрядах 0 - 5 второго слова.

В полях типа *signed*крайний левый бит является знаковым.

Поля используются для упаковки значений нескольких переменных в одно машинное слово с целью экономии памяти. Они не могут быть массивами и не имеют адресов, поэтому к ним нельзя применять унарную операцию &.

*struct Student{*

*char sirname[80];*

*char group[20];*

*int age;*

*int marks[5];*

*};*

*Student one; //структура*

*Studentmass[10]; //массив структур*

*Student \*p; //указатель на структуру*

*union r {intir; float fr; char cr;} z;*

Здесь ir имеет размер 4 байта, fr - 4 байта, cr - 1 байт. Размер переменной z будет равен размеру самого большого из трех приведенных типов (т.е. 4 байтам). В один и тот же момент времени z может иметь значение только одной из переменных ir, fr или cr.

***Задание:***

1. Написать программу, которая читает данные о студентах (фамилия, группа, возраст, успеваемость) из файла и выводит данные по самому успевающему студенту. Использовать массив структур Student для решения задачи.

***Задание на дом*:** лабораторная работа №9

# Практическое занятие № 10

***Тема:****Символьные указатели и строки символов. Массивы указателей. Указатели на указатели.*

* Многие строковые операции в С выполняются с помощью указателей, поскольку доступ к строкам осуществляется последовательно.
* Например, здесь показана версия стандартной библиотечной функции strcmp(), использующей указатели:

*/\* Использованиеуказателей. \*/*

*intstrcmp (const char \*s1, const char \*s2)*

*{*

*while(\*s1) if(\*s1-\*s2) return \*s1-\*s2;*

*else {*

*s1++;*

*s2++;*

*}*

*return 0; /\* равенство \*/*

*}*

* Надо помнить, что все строки в С оканчиваются нулевым символом, который интерпретируется как ложь. Следовательно, оператор *while* (\*s1) выдает истину, пока не достигнет конца строки. *strcmp*() возвращает 0, если s1 равно s2. Она возвращает число меньше 0, если s1 меньше s2. Иначе возвращает число больше нуля.
* Большинство строковых функций похожи по способу работы на *strcmp*(). Особенно там, где используется контролирующий цикл. Использование указателей быстрее, более эффективно и часто легче для понимания, чем использование индексации массива.
* Функция *strcpy* копирует свой второй аргумент — строку в свой первый аргумент — массив символов. Программист должен быть уверен, что массив достаточного размера, чтобы хранить строку и ее завершающий нулевой символ.
* Функция *strncpy* эквивалентна *strсpy*, за исключением того, что в вызове *strncpy* указывается количество символов, которое должно быть скопировано из строки в массив. Завершающий нулевой символ будет копироваться, только если количество символов, которое должно быть скопировано, по крайней мере, на один больше длины строки.
* Функция *strcat* добавляет свой второй аргумент - строку к своему первому аргументу - массиву символов, содержащему строку. Первый символ второго аргумента замещает нулевой символ ('\0'), который завершал строку в первом аргументе. Программист должен быть уверен, что массив, используемый для хранения первой строки, достаточно великдля того, чтобы хранить комбинацию первой строки и второй строки.
* Функция *strncat* добавляет указанное количество символов из второй строки в первую строку. К результату добавляется завершающий нулевой символ.
* Функция strcmp сравнивает символ за символом строку в своем первом аргументе со строкой в своем втором аргументом. Функция возвращает 0, если строки равны, отрицательное значение, если первая строка меньше, чем вторая, и положительное значение, если первая строка больше, чем вторая.
* Функция *strncmp* эквивалентна *strcmp*, за исключением того, что *strncmp* сравнивает указанное количество символов. Если количество символов одной из строк меньше чем указанное количество символов, *strncmp* сравнивает символы до тех пор, пока не встретится нулевой символ в более короткой строке.
* Последовательные вызовы функции *strtok* разбивают строку на лексемы, разделенные символами, содержащимися во втором аргументе — строке. При первом вызове в качестве первого аргумента передается строка, разбиваемая на лексемы, а при последующих вызовах, продолжающих разбиение на лексемы той же самой строки, в качестве первого аргумента передается NULL. При каждом вызове возвращается указатель на текущую лексему. Если при вызове *strtok* лексем больше нет, возвращается NULL.
* Функция *strlen*получает в качестве аргумента строку, а возвращает количество символов в строке; завершающий нулевой символ в длину строки не входит.

***Задание***

1. Написать программу, которая выделяет из заданной строки подстроку заданной длины, начиная с заданной позиции. (Функция *substr*()).Осуществлять проверку выхода за размерность строки. При обращении к элементам массива использовать обращение по указателю (выражения вида \*p, p++, \*p++ …).

***Задание на дом***: лабораторная работа №10

# Практическое занятие № 11

***Тема****: Функции. Определение функции. Вызов функции. Прототипы функции.Передача параметров в функцию.*

* Лучший способ разработки и поддержки большой программы - разделить ее на несколько меньших модулей, каждый из которых более управляем, чем исходная программа. Модули пишутся на C++ в виде функций и классов.
* Функция активизируется посредством вызова функции. В вызове указывается имя функции и передается информация (в виде аргументов), которая нужна вызываемой функции для выполнения ее задачи.
* Функции обычно активизируются в программе написанием имени функции, за которым следуют аргументы функции в круглых скобках.
* Каждый аргумент функции может быть константой, переменной или выражением.
* Локальная переменная известна только в описании данной функции. Функции не знают детали реализации другой функции (включая локальные переменные).
* Общий формат описания функции:

*Тип-возвращаемого-значения имя-функции (список-параметров) {объявления и операторы}*

* Тип-возвращаемого-значения устанавливает тип значения, возвращаемого в вызывающую функцию. Если функция не возвращает значение, тип-возвращаемого-значения объявляется как void. Имя-функции - любой правильно написанный идентификатор. Список-параметров - написанный через запятые список, содержащий объявления переменных, которые будут переданы функции. Если функция не предусматривает передачу в нее никаких значений, список-параметров объявляется как void. Тело-функции - набор объявлений и операторов, которые составляют функцию.
* Аргументы, передаваемые функции, должны быть согласованы по количеству, типу и порядку следования с параметрами в описании функции.
* Прототип функции объявляет тип возвращаемого значения функции, количество, типы и порядок следования параметров, передаваемых в функцию.
* Прототипы функций дают возможность компилятору проверить, правильно ли вызывается функция.

Передача параметров в функцию.

* *Передача параметра по значению.* Передача параметра по значению означает что вызывающая функция копирует в память, доступную вызываемой, (обычно стек) непосредственное значение. Изменение копии переменной, соответственно, оригинал не затрагивает.
* *Передача параметра по адресу.* Если необходимо именно изменить переменную из внешней, по отношению к вызываемой функции, области видимости, можно копировать адрес переменной, подлежащей изменению. Соответственно при вызове функции приходится использовать операцию взятия адреса.
* *Передача параметра по ссылке* означает, что копируется не само значение, а адрес исходной переменной (как в случае передачи параметра по адресу), однако синтаксис используется такой, чтобы программисту не приходилось использовать операцию разыменования и он мог иметь дело непосредственно со значением, хранящимся по этому адресу (как в случае передачи параметра по значению).

*structDrob{*

*intm\_celoe*

*intm\_chis;*

*int m\_ znamen;*

*};*

*Drobysum(Droby& a, Droby& b);*

*Drobysub(Droby&a,Droby& b);*

*Drobymul(Droby&a,Droby& b);*

*Droby div(Droby& a,Droby& b);*

*void show(Droby& a);*

***Задание:***

1. Написать программу, которая позволяет выбирать и совершать операции сложения, вычитания, умножения или деления между двумя дробями (целая часть, числитель и знаменатель) и выводить результат операции на экран.

Главный интерфейс программы:

***Введите первую дробь (Цел.Числ. Знам.)***

***Введите вторую дробь (Цел.Числ. Знам.)***

Далее в бесконечном цикле выдаются запросы:

1 - ***Сложение***

2 ***- Вычитание***

3 -***Умножение***

4 - ***Деление***

ESC-***Выход***

**Результат:** *(в удобном для понимания виде: 1 2/3 или 2/3 или3 или 0, но не 0 1/2)*

***Задание на дом:***лабораторная работа № 11

# Практическое занятие № 12

***Тема:****Области видимости. Виды областей видимости. Файловая область видимости. Операция глобальной области видимости. Объявления внешних переменных. Статические глобальные переменные и функции. Локальная область видимости. Автоматические локальные переменные.Пространство имен.*

* Каждый идентификатор переменной характеризуется классом памяти, областью действия и компоновкой.
* C++ обеспечивает четыре спецификации класса памяти: ***auto****,* ***register****,* ***extern*** *и* ***static***.
* Класс памяти идентификатора определяет время жизни этого идентификатора в памяти.
* Область действия идентификатора определяет, где в программе можно ссылаться на этот идентификатор.
* Компоновка идентификатора определяет для программы с многими исходными файлами, известен ли идентификатор только в текущем исходном файле или в любом исходном файле с соответствующим объявлением.
* Переменные автоматического класса памяти создаются при вхождении в блок, в котором они определены, существуют лишь пока этот блок активен, и уничтожаются при выходе из блока.
* Спецификатор класса памяти ***register*** может быть помещен перед объявлением автоматической переменной, чтобы указать компилятору разместить переменную в одном из высокоскоростных аппаратных регистров компьютера. Компьютер может проигнорировать объявление ***register***. Ключевое слово ***register*** можно использовать только для переменных автоматического класса памяти.
* Ключевые слова ***extern*** и ***static*** используются, чтобы объявить идентификаторы переменных и функций статического класса памяти.
* Переменные статического класса памяти размещаются и получают начальные значения в начале выполнения программы.
* Глобальные переменные создаются путем помещения их объявления вне какого-либо описания функции, и они сохраняют свои значения в течение всего времени выполнения программы.
* Локальные переменные, объявленные как ***static***, сохраняют свои значения после выхода из функции, в которой они объявлены.
* Все численные переменные статического класса памяти получают нулевые начальные значения, если программист не присвоил им явно другие начальные значения.
* Существуют четыре области действия идентификаторов: область действия функция, область действия файл, область действия блок и область действия прототип функции.
* Метки являются единственными идентификаторами с областью действия функция. Метки можно использовать всюду внутри функции, в которой они находятся, но на них нельзя ссылаться вне тела функции.
* Идентификатор, объявленный вне любой функции, имеет областью действия файл. Такой идентификатор «известен» с момента его объявления до конца файла.
* Идентификаторы, объявленные внутри блока, имеют областью действия блок. Область действия блока заканчивается завершающей правой фигурной скобкой (}).
* Локальные переменные, объявленные в начале функции, имеют областью действия блок подобно параметрам функции, которые считаются локальными переменными функции.
* Любой блок может содержать объявления переменных. Если блоки вложены и идентификатор во внешнем блоки имеет такое же имя, как идентификатор во внутреннем блоке, идентификатор во внешнем блоки «невидим» (скрыт) до тех пор, пока не завершится внутренний блок.
* Единственными идентификаторами, имеющими областью действия прототип функции, являются те, которые использованы в списке параметров прототипа функции. Идентификаторы, использованные в прототипе функции, можно повторно использовать в других местах программы без опасений возникновения неопределенности.
* Пространство имен (***namespace***) является фундаментальной концепцией C++.
* Пространство имен - это группа имен, в которой имена не совпадают. Исключением являются имена перегружаемых функций и переменные с различными областями действия.
* Имена в различных пространствах имен не конфликтуют.
* C++ также позволяет определить пространства имен при помощи ключевого слова ***namespace*** . Такие пространства имен вводятся для снижения вероятности конфликта имен и полезны в случае использования имен из нескольких различных библиотек.
* Спецификатор ***namespace*** определяет пространство имен функций, классов и переменных, находящихся в отдельной области видимости. Вообще ***namespace*** определяет группу символов, представляющих собой названия или имена функций, классов или переменных.

Синтаксис:

*namespacename*

*{*

*// объявления и определения имен*

*}*

Аргументом спецификатора ***namespace*** (то есть name ) является идентификатор пространства имен.

* Чтобы обратиться к чему-нибудь в пространстве*namespace\_name*, достаточно обратиться непосредственно к члену, используя оператор определения области видимости (::): *namespace\_name::member* . Однако, при многократном использовании члена *member* такая запись становится слишком громоздкой.
* Оператор using заставляет компилятор признавать дальнейшее использование этому члену пространства *namespace\_name* без дополнительного определения имени этого пространства: *usingnamespace\_name::member*;
* Оператор*usingnamespace* заставляет компилятор признавать все члены пространства имен *namespace\_name: usingnamespacenamespace\_name*;

***Задание:***

1. Функция f1() описана в файле a.cpp, функция f2() в файле b.cpp и функция f3() в файле c.cpp и main, описана в d.cpp. Функция f1() и f2() проверяют состояние глобальной переменной flag, определение которой дано в d.cpp и изменяют ее состояние на противоположное. Функция f3() содержит статическую переменную see\_stat, которая наращивается на единицу при каждом вызове функции (++). Написать программу, которая вызывает выполнение функций f1(), f2() и f3() многократно и выводит на экран текущее состояние переменной flag и значение статической переменной see\_stat. Используйте проект для создания рабочего модуля программы. Объясните разницу между определением и объявлением переменной. Дайте определение - время жизни и область видимости локальных, глобальных и статических переменных.

***Задание на дом*:** лабораторная работа №12

# Практическое занятие № 13

***Тема:*** *Параметры функции main и функции файлового ввода и вывода*

В программы на языке Си можно передавать некоторые аргументы. Когда производится обращение к main( ), ей передаются три параметра. Допустимыследующиеописания:

*main( )*

*main(intargc)*

*main(intargc, char \*argv[ ] )*

*main(intargc, char \*argv[ ], char \*env[ ] )*

* Аргумент argc типа int содержит в себе количество аргументов командной строки, передаваемых в программу.
* Аргумент argv типа char \*\* - массив указателей на символьные строки(в одной строке - один аргумент). Каждый элемент массива указывает на аргументы командной строки. Один параметр отделяется от другого пробелами.

*argv[0] - полное имя запущенной программы*

*argv[1] - первая строка записанная после имени программы*

*argv[2] - вторая строка записанная после имени программы*

*argv[argc-1] - последняя строка записанная после имени программы*

*argv[argc] - NULL*

* Третий аргумент env - так же как и argv является массивом указателей на символьные строки, используется для доступа к параметрам операционной системы (к переменным окружения).
* Если необходимо в качестве параметра иметь строку, содержащую пробелы, то ее надо заключить в двойные кавычки. Если аргументом является число, то оно рассматривается как строка. Для работы с ним, как с числом необходимо его преобразовать, используя соответствующую функцию.

Пример:

#*include "stdio.h"*

*#include "stdlib.h"*

*int main(intargc, char \*argv[])*

*{*

*int i;*

*printf("Количество аргументов командной строки %d \n", argc);*

*printf("Аргументы командной строки:\n);*

*for (i = 0; i<argc; i++) printf("%s\n", argv[i]);*

*return 0;*

*}.*

* Файловая система ANSI С предоставляет две функции, позволяющие читать и писать блоки данных - fread() и fwrite(). Они имеют следующие прототипы:

*size\_tfread(void \*буфер, size\_tчисло\_байтуsize\_tобъем, FILE \*fp);*

*size\_tfwrite(const void \*буфер, size\_tчисло\_байт, size\_tобъем, FILE \*fp);*

* В случае fread() буфер - это указатель на область памяти, которая получает данные из файла. В случае fwrite() буфер - это указатель на информацию, записываемую в файл. Длина каждого элемента в байтах определяется в *число\_байт*. Аргумент объем определяет, сколько элементов (каждый длиной *число\_байт*) будет прочитано или записано. Наконец, fp - это файловый указатель на ранее открытый поток.
* Функция fread() возвращает число прочитанных элементов. Данное значение может быть меньше, чем объем, если был достигнут конец файла или произошла ошибка. Функция fwrite() возвращает число записанных элементов. Данное значение равно объем, если не возникла ошибка.
* Если файл был открыт для двоичных данных, то fread() и fwrite() могут читать и писать любой тип информации. Например, следующая программа записывает float в файл:

*#include<stdio.h>*

*intmain(void)*

*{*

*FILE \*fp;*

*float f = 12.23;*

*if((fp=fopen("test", "wb"))==NULL) {*

*printf("Cannot open file.");*

*return 1;*

*}*

*fwrite(&f, sizeof (float), 1, fp);*

*fclose (fp);*

*return 0;*

*}*

* Использование *fread*() и *fwrite*() для чтения или записи сложных данных более эффективно, чем использование повторяющихся вызовов *getc*() и *putc*().

***Задание:***

1. Написать программу копирования файлов (команда copy операционной системы) передавая имя входного и выходного файла с командной строки. Организовать проверку наличия 2-х входных параметров (имя входного и выходного файла и в случае ошибки выдавать небольшой help по использованию программы).

***Задание на дом*:** лабораторная работа №13

# Практическое занятие № 14

***Тема:*** *Указатели на функции*

В языке Си сама функция не может быть значением переменной, но можно определить указатель на функцию. С ним уже можно обращаться как с переменной: передавать его другим функциям, помещать в массивы и т.п.

Типичным применением указателей на функции являются так называемые системы, управляемые меню. Указатели на функции используются для вызова функций, выбираемых с помощью отдельных разделов меню.

***Задание:***

1. Написать программу, которая читает данные о студентах из файла и выводит на экран отсортированные данные по студентам по полю, указанному по запросу (Фамилия, возраст, группа, успеваемость). Использовать массив структур Student и указатели на функции(в функцию сортировки в качестве параметра передавать указатель на функцию сравнения поля двух объектов типа Student).
2. Что означают следующие описания:

* *inti;*
* *int \*pi;*
* *int \*api[3];*
* *int (\*pai)[3];*
* *int \*\*pai;*
* *int fi();*
* *int \*fpi();*
* *int (\*pfi)();*
* *int \*(\*pfpi)();*
* *int (\*pfpfi())();*
* *int (\*fai())[3];*
* *int (\*apfi[3])();*
* *int (\*f())();*
* *int (\*f[3])();*
* *int\*(\*(\*(\*g)())[5])();*
* *char \* (\*&(\*index)())[10];*

***Задание на дом:***Лабораторная работа №14

# Практическое занятие № 15

***Тема:****Функции. Функции с переменным числом параметров*

* Язык С позволяет писать функции с переменным числом параметров.
* Переменный список параметров задается в заголовке функции многоточием: int f(…)
* Поскольку имена параметров отсутствуют доступ можно осуществить только одним способом – косвенным, используя указатель.
* Все параметры при вызове помещаются в стек. Если мы каким-то образом установим указатель на начало списка параметров в стеке, то, манипулируя с указателем, мы, в принципе, можем «достать» все параметры!
* Список параметров совсем пустой быть не может, должен быть прописан хотя бы один явный параметр, адрес которого мы можем получить при выполнении программы. Заголовок такой функции может выглядеть так: int f(int k...).
* Есть одно обстоятельство, которое ограничивает применение таких функций: при написании функции с переменным числом параметров помимо алгоритма обработки программист должен разрабатывать и алгоритм доступа к параметрам. Так что список необъявленных параметров не может быть совсем уж произвольным – в языке C не существует универсальных средств распознавания элементов этого списка. Это же означает, что передача аргумента не того типа, который задумывался, или не тем способом, который подразумевался при разработке, приведет к катастрофическим последствиям – компилятор-то ничего не проверяет.
* Это же означает, что программист при разработке функции с переменным числом параметров должен отчетливо себе представлять типы аргументов, которые будет обрабатывать функция.

Примеры библиотечных функций с переменным числом параметров: *printf(), scanf();*

***Задание*:**

1. Написать программу вызова функций с различным числом параметров – sum(), sum1(), sum2() и вывода на экран результата работы этих функций.

intsum(intn,…) – Целая переменная - счетчик, затем последовательность целых переменных. Функция возвращает сумму переменных.

intsum1(intn,…) . Последовательность целых переменных, ограниченная переменной со значением 0. Функция возвращает сумму значений параметров.

doublesum2(int n, …) – функция нахождения суммы чисел разного типа. Первый параметр - форматная строка, в которой каждая цифра обозначает тип очередного параметра: 1-целое, 2-длинное целое, 3-вещественнное, 0-конец последовательности. Функция возвращает сумму значений параметров.

***Задание на дом*:** лабораторная работа №15

# Практическое занятие № 16

***Тема:****Перегрузка (совместное использование) имен функций. Объявление перегруженной функции. Сопоставление параметров. Точное сопоставление. Расширение типа.*

* Возможно определение нескольких функций с одинаковыми именами, но разными типами параметров. Эти функции называются перегруженными. При вызове перегруженной функции компилятор выбирает соответствующую функцию, анализируя количество и тип аргументов в вызове.
* Перегруженные функции могут иметь разные или одинаковые типы возвращаемых значений и обязательно должны иметь разные списки параметров. Две функции, отличающиеся только типами возвращаемых значений, вызовут ошибку компиляции.

Примерыперегруженныхфункций:

*int min(int \*a, int size);*

*longmin(long \*a, int size);*

*float min(float \*a, int size);*

*double min(double \*a, int size);*

*long sum(int \*a, int size);*

*long sum(long \*a, int size);*

*float sum(float \*a, int size);*

*double sum(double \*a, int size);*

***Задание:***

1. Написать программу, которая определяет минимальное значение и сумму элементов массива для массивов различного типа (int, long, float, double) из 100 элементов, используя для этого аппарат перегруженных функций. Заполнять элементы массива случайными числами и выводить их эначения на экран.

***Задание на дом:***Лабораторная работа № 16

# Практическое занятие № 17

*Задание :*

1. Решить задачу из занятия 16 на основе шаблонных функций. Добавить шаблонные функции нахождения максимума, среднего значения массива.

# Практическое занятие №18

Семинар по вопросам всего курса( см выше)

**Литература**

***основная литература:***

1. Павловская Т. C/C++. Программирование на языке высокого уровня: Учебник для вузов, СПб. Питер. 2009, 464 стр.
2. Подбельский В.В., Фомин С.С., Программирование на языке СИ. М. Финансы и статистика. 2008. 600 с.
3. Прата С. Язык программирования C++. Лекции и упражнения, Вильямс, 1184 стр.
4. Керниган Б. У., Ритчи Д. М., Язык программирования C (Си), 2-е издание, Вильямс, 2008, 304 стр.
5. Шилдт Г. C++: базовый курс, 3-е издание; Вильямс 2008, 624 стр.
6. Шилдт Г. Полный справочник по C. Язык программирования Си. 4-е издание, Вильямс,
7. Дейтел Пол Дж., Дейтел Харви. Как программировать на С++. Бином. 2009. 800с.

***б) дополнительная литература:***

1. Страуструп Б. Язык программирования С++. Специальное издание. Бином. 2008. 1104с.
2. Франка П.C++: учебный курс. СПб. Питер, 2003. 521 с.
3. Эккель Б. Философия С++. Введение в стандартный С++. СПб. Питер. 2004. 572 с.
4. Эккель Б., Эллисон Ч. Философия С++. Практическое программирование. СПб. Питер. 2004. 572 с.
5. Мейерс С. Эффективное использование С++. 50 рекомендаций по улучшению ваших программ и проектов. М. ДМК Пресс. 2006. 240с.
6. Мейерс С. Эффективное использование С++. 55 верных способов улучшить структуру и код ваших программ. М. ДМК Пресс. 2006. 300с.
7. Голуб Ален И. Веревкадостаточной длины, чтобы…«выстрелитьсебе в ногу». Правила программирования на Си и Си++. Москва. Бином. 2001. 240 с

***в) программное обеспечение и Интернет-ресурсы***

1. <http://www.codeblocks.org/>Code::Blocks — свободная кроссплатформенная среда разработки программ на языке С++. setlocale(LC\_CTYPE,"Russian");
2. Microsoft Developer Network. <http://msdn.microsoft.com/ru-ru/library/>
3. Программирование на C и C++ . <http://www.c-cpp.ru/books/>