Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра интеллектуальных информационных технологий

Ю.Г. ПРИХОДЬКО, С.А. САМОДУМКИН, О.Е. ЕЛИСЕЕВА

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по курсу

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

для студентов специальности

"Искусственный интеллект"

В 3-х частях

ЧАСТЬ 2

Минск 2001

УДК 681.3 (075.8)
ББК 32.973 Я 73
П 77

Приходько Ю.Г. и др.

Лабораторный практикум по курсу "Основы алгоритмизации и
П 77 программирования" для студентов специальности "Искусственный интеллект" /
Ю.Г. Приходько, С.А. Самодумкин, О.Е. Елисеева. В 3ч. Ч.2.–Мн.:БГУИР,2001. - с.

ISBN

В работе приводятся краткие теоретические сведения и индивидуальные задания к лабораторным работам по основам программирования на языке СИ. Рассматриваются темы разработка программ на алгоритмическом языке Си, программирование ветвящихся процессов, программирование циклических структур и организация подпрограмм. Практикум содержит описание заданий к лабораторным работам по курсу “Основы алгоритмизации и программирования”, который читается студентам специальности “Искусственный интеллект”.

УДК 681.3 (075.8)
ББК 32.973 Я 73

ISBN

Содержание

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: знакомство с интегрированными системами программирования на языке Си, разработка, компиляция и отладка простейших С-программ.

Цель: изучить интегрированную среду Borland C, научиться создавать проекта программы, компилировать и отлаживать простейшей С-программы

Краткие теоретические сведения

3.1. Интегрированная среда программирования Borland C++

3.1.1. Главное меню

Среди характерных особенностей интегрированной среды Borland C++ можно выделить мощную оконную систему с возможностью передвижения и изменения размера окон, поддержку манипулятора "мышь", многооконный текстовый редактор, окна диалогов (dialogs boxes), различные меню. После запуска на исполнение файла IDE (BC.EXE) на экране отображается основное окно IDE.

Верхняя строка окна - это главное меню (the menu bar). Оно позпозволяет обратиться к подсистемам (подменю):системному меню (=), меню файловой системы(File),меню редактирования (Edit), меню поиска и замещения (Search), меню управления исполнением программ (Run), меню компиляции программ (Compile), меню управления проектом (Project), меню встроенного отладчика программ (Debug), меню опций (Options), меню управления окнами (Window) и меню помощи (Help).

Нижняя строка экрана отведена под так называемую строку состояния (the status line), где выделены назначения "горячих" клавиш (hot keys), воспринимаемых на данном этапе работы. Например, часто строка состояния имеет такой вид: F1 - Help, F2 - Save, F3 - Open, Alt-F9 - Compile, F9 - Make, F10 - Menu.

Выбрать любую из команд главного меню можно одним из трех способов:

1) нажать клавишу F10 ("войти" в главное меню) и с помощью клавиш со стрелками выбрать необходимую команду. Она подсвечивается другим цветом. Для выполнения команды нажать клавишу ENTER. Вариация метода: войти в главное меню и нажать символьную клавишу, совпадающую с выделенной в команде литерой. Это, как правило, начальная литера слова, выделяемая другим цветом;

2) установить курсор манипулятора "мышь" на любое ключевое слово меню и нажать левую кнопку манипулятора;

3) использовать "горячие" клавиши. Одновременное нажатие клавиши Alt и "горячей" клавиши активизирует соответствующую команду. Например, команду Compile можно выбрать, нажимая Alt-C. Исключением из этого правила является переход к системному меню, в этом случае "горячей" клавишей для вызова будет Alt-SPACEBAR.

Выбор команды часто связан с появлением на экране еще одного уточняющего меню. Выбор из подменю аналогичен работе с главным меню.

3.1.2. Управление окнами

Одной из наиболее примечательных особенностей IDE является мощная оконная система. С. Окно - это ограниченная рамкой область экрана. Его можно открыть, переместить, покрыть другими окнами, изменить размеры, закрыть. Активное окно обозначается двойной линией, и в текущий момент может быть только одно активное окно. Каждое окно имеет заголовок и номер, показанные в его верхней строке. Для активного окна там же расположены два управляющих поля, заключенных в квадратные скобки: поле справа используется для быстрого раскрытая окна "мышью" на полный экран, а поле слева - для закрытия окна, Многие окна для управления положением текста имеют по правой и нижней границам вертикальную и горизонтальную полосы прокрутки (скроллинга). Поля изменений размеров окна - это символы нижних углов рамки окна

Операции с окнами могут выполняться тремя способами:

1) через команды главного меню Window;

2) с помощью манипулятора "мышь";

3) "горячими" клавишами;

Окна могут располагаться либо с наложением друг на друга (Cascade), либо в одной плоскости (Tile).

Закрыть можно только активное окно. Для этого либо выбирается Main Menu-Window-Close, либо нажимается "горячая" клавиша Alt-F3, либо "мышью".

Активное окно изображается всегда поверх других окон. Переключение окон с помощью меню или клавиатуры выполняется так. Выбирается
Main Menu Window-List... или нажимается "горячая" клавиша Alt-0. Открывается окно диалога, в котором располагаются заголовки всех окон, в том числе и ранее закрытых. Выбрав "мышью" или клавиатурой нужное окно, можно сделать его активным. Для первых девяти окон быстрая активизация возможна нажатием Alt-номер окна. Таких ограничений нет при активизации окон с помощью манипулятора "мышь". Если на экране отображена хотя бы небольшая часть необходимого окна, достаточно в эту область установить "мышь" и нажать ее кнопку, чтобы окно стало активным.

Изменение размера и (или) перемещение активного окна выполняется так:

1) выбирается Main Menu-Window-Size/Move или нажимается "горячая" клавиша Ctrl-F5;

2) клавиши со стрелками будут перемещать активное окно в любое место в пределах экрана; для изменения правой или нижней страницы окна используются клавиши со стрелками, но одновременно удерживается нажатой клавиша Shift;

3) новый размер и (или) положение окна фиксируется нажатием клавиши ENTER.

Те же самые действия могут быть выполнены и с помощью "мыши". Чтобы изменить размеры окна, необходимо установить "мышь" на правый или левый нижний угол рамки и, удерживая нажатой кнопку манипулятора, перемешать "мышь". После этого кнопку следует отпустить. Те же действия в случае, когда "мышь" установлена на полосу заголовка либо верхнюю рамку окна, приведут к перемещению окна по экрану монитора.

Активное окно может быть раскрыто на весь экран либо выбором Main Menu-Window-Zoom, либо нажатием клавиши F5, либо выбором "мышью". Повторение этих действий возвращает окно к прежним размерам.

Меню Main Menu-Window позволяет выбрать несколько специализированных окон. Окно сообщений (Message) используется для активизации окна, в которое выводятся все сообщения об ошибках и предупреждения, генерируемые при компиляции и (или) компоновке программы.

Для просмотра результатов выполнения программы используется переключение в окно вывода (Output). "Горячая" клавиша для этих действий - Alt-F5. Возврат в среду происходит при нажатии любой клавиши.

Окно Output дает возможность просмотреть результаты работы программы, если вывод на экран выполняется в текстовом режиме. Для просмотра результатов работы программы как в текстовом, так и графическом режимах, следует активизировать окно экрана пользователя (User Screen).

Два специализированных окна Watch и Register используются при отладке программ. Окна Project и Project Notes выделены для упрощения работы с файлами проектов, которые используются при многофайловой компиляции.

3.1.3. Окна диалогов. Подсистема работы с файлами

Одной из разновидностей окон в IDE выступают так называемые окна диалогов (dialog boxes). Они предназначены для управления работой различных команд IDE. С их помощью можно уточнить начальные условия и режимы работы команд. Для всех строк меню, где после наименования команды указано обозначение "...", в системе есть свое окно диалога. Например, при выборе команды Files открывается меню файловой системы, в котором команды Open..., Save as..., Change dir... и Get info... имеют еще и окна диалога.

Переход от одной группы элементов окна диалога к другой (активизация элемента) выполняется нажатием клавиши Tab (движение

вперед) или Shift-Tab (движение назад), либо выбирается манипулятором "мышь". Активный элемент высвечивается другим цветом. "Нажатие кнопки" управления выполняется одним из трех способов:

1) клавишами Tab (Shift-Tab) активизируется нужная кнопка и после этого нажимается ENTER;

2) нажимается "горячая" клавиша, совпадающая с высвеченной в названии кнопки литерой;

3) нужная кнопка выбирается манипулятором "мышь". Количество и функции кнопок зависят от окна диалога. Для окна диалога Open нажатие кнопки Open загружает текстовый файл в новое активное окно редактора. Имя загружаемого файла отображается в нижнем левом углу окна диалога. Нажатие кнопки Replace за1ружает файл в текущее активное окно, не создавая новых окон. Кнопки Cancel (Отмена) и Help (Помощь) присутствуют во всех окнах диалога. Первая из них закрывает окно, ничего не изменяя в системе. Другая выдает окно информации об активном элементе окна диалога и выполняемых им функциях.

Поле Name, когда оно активно, используется для набора имени файла или имени файла с символами '*' и '?'. В этом случае имя используется как шаблон для фильтрации файлов в окно Files. Если надо изменить шаблон их фильтрации, задав другое расширение, можно использовать поле, помеченное символом ↓. Назначение этого символа во всех окнах диалога - обращение к окну истории.

Поле списка - это высвеченный другим цветом прямоугольник в центре окна диалога. В окне Open - это список файлов и директорией, позволяющий выбрать нужный файл или перейти в другой директорий, но только в пределах текущего накопителя.

Помимо команды Open, подсистема работы с файлами содержит и рад других команд. Прежде всего - это команда New, "начинающая" редактирование нового текстового файла. Для него будет открыто новое окно, а сам файл получит имя NONAMEOO.C имя NONAMEOO.CPP.

Если необходимо переименовать файл, изменить его директорий или букву накопителя, на котором хранится файл, используется команда Save as...

Команда Save сохраняет файл в активном окне редактора текстов. Если это файл с именем NONAMEOO.* или подобным ему, IDE открывает окно диалога, позволяющее задать другое имя "нового" файла.

Команда Save all выполняет сохранение модифицированных файлов во всех окнах редактора текстов.

Для задания текущих накопителя и (или) директория, используемых по умолчанию при загрузке и сохранении файлов, служит команда Change dir....

Окно, помеченное Directory Tree, отображает все директории ниже текущего в файловой иерархии. Текущий директорий показан сразу же после строки Drives и отмечен особым символом. Задать нужный директорий и (или) накопитель можно двумя способами:

1) ввести маршрут с клавиатуры в окне ввода Directory Name и нажать клавишу ENTER;

2) используя дерево файловой иерархии, выбрать нужный директорий и нажать ENTER.

Нажатие кнопки управления OK или клавиши ESC завершает процедуру смены директория и (или) накопителя.

Строка Drives дает возможность, не прибегая к клавиатурному вводу, задать букву накопителя. Если в иерархии высвечена строка Drives, двойное нажатие кнопки "мыши" или нажатие ENTER отображает в окне Directory Tree список всех букв и дает возможность выбрать нужный накопитель.

Если возникает необходимость вернуться к прежним значениям, "нажимается" кнопка Revert.

Команда DOS shell позволяет временно выйти из IDE в MS-DOS. Для возврата в IDE выполняется команда MS-DOS Exit.

Завершение работы интегрированной среды выполняет команда Quit. Если в этот момент есть файлы, в которых сделаны изменения, и они не сохранены на диске, IDE выдает приглашение подтвердить выход без сохранения сделанных изменений.

3.1.5. Многооконный текстовый редактор

Переключение в режим редактирования выполняется либо явно через главное меню, либо автоматически при выборе команды New в меню File, при открытии файла, при работе с ошибками, выявленными на этапе компиляции.

Традиционно команды редактора разбивают на группы:

1) команды перемещения курсора;
2) команды вставки и удаления;
3) команды работы с блоками;
4) команды работы с блоками в разных окнах;
5) команды поиска и замены;
6) другие команды.

Перемещение курсора можно выполнить не только клавишами, но и с помощью манипулятора "мышь". На экране всегда присутствуют два независимых курсора: управляемый клавиатурой текстовый и управляемый "мышью". Если нажать левую кнопку "мыши", текстовый курсор переместится в позицию курсора "мыши". Для скроллинга текста "мышью" используются вертикальная и горизонтальная полосы скроллинга.

Целая группа команд редактора Turbo C++ предназначена для работы с блоками текста. Выделенный в блок текст на экране отображается ярким цветом.

Для обслуживания межоконного переноса блоков используется специальная область для накопления блоков текста (Clipboard). Для записи отмеченного блока текста в Clipboard либо выполняется команда Main Menu-Edit-Copy, либо просто нажимается "горячая" клавиша Ctrl-Ins. Блоки в Clipboard накапливаются; последний перенесенный туда блок остается выделенным. Просмотр текущего содержимого Clipboard требует открытия окна командой Main Menu-Edit-Show Clipboard.

Интегрированная среда позволяет скопировать блок текста и из окна, открываемого Help-системой. Для этого выделяется нужный блок текста в окне Help, который копируется в Clipboard. Отсюда его можно вставить в любое другое окно. Рассмотренная возможность исключительно полезна при изучении языка Си, так как в окнах Help описания библиотечных функций приводятся примеры текстов программ. Копирование этих

примеров, их компиляция и запуск на выполнение - прекрасное средство самообразования.

Если блок необходимо "вырезать" из активного окна редактора и записать его в Clipboard, используется команда Cut. Для извлечения блока из Clipboard используется либо команда Main Menu-Edit-Past, либо "горячая" клавиша Shift-Ins.

Текстовый редактор IDE имеет удобные средства для поиска нужного фрагмента текста и его замещения другим фрагментом. Управление режимами поиска и замены осуществляется через меню Search, включающее команды Find..., Replace..., Search again, Go to line number..., Previous error, Next error и Locate function....

Для выполнения операции необходимо ввести строку текста, которую требуется отыскать в активном окне, и задать режимы поиска. Ввод текста выполняется в поле Text to Find. Выбор "мышью" поля, помеченного стрелкой, или нажатие клавиши Down откроет окно истории, где будут помещены строки, которые задавались при предыдущих операциях поиска.

В качестве примера рассмотрим поиск слова "#include", считая строчные и прописные буквы разными символами (Case sensitive); поиск начать с текущей позиции курсора (From Cursor), а не с самого начала (Entire Scope); продвигаться по тексту вперед до конца файла (Forward), а не назад (Backward); вести поиск по всему файлу (Global), а не по выделенному тексту (Selected text). Чтобы перейти к выполнению операции поиска, следует нажать либо клавишу ENTER, либо кнопку OK. Если слово "#include" найдено в тексте, то на экране оно будет выделено другим цветом. чтобы продолжить поиск с заданными условиями, следует выбрать команду Main Menu -Search -Search again или нажать "горячую" клавишу Ctrl-L.

Перечислим другие опции поиска:

Whole words only - осуществляется поиск строки текста, имеющей с обоих концов пробелы или другие разделители слов текста;

Regular expression - поиск строки ведется по шаблону, заданному в поле Text to Find. Шаблон может включать в себя такие символы: ^, $, .,*,+, [ ] и \. Они означают следующее:

. (точка) в строке шаблона соответствует любому символу строки в просматриваемом тексте. Например, b.t соответствует bit, bot, bat;

^ ("крышечка") в начале строки шаблона соответствует любой совокупности символов в начале слова просматриваемого текста. Например, ^t соответствует at, cat, must;

$ (знак доллара) в конце строки шаблона соответствует любой совокупности символов в конце слова просматриваемого текста. Например, b$ соответствует bo, bod, bort;

* (звездочка) после символа соответствует любому количеству совпадений с символом (в том числе и нулевому). Например, bo* соответствует bot, b, boo, be;

+ (плюс) после символа соответствует любому количеству совпадений (но не нулевому) с символом. Например, bo+ соответствует bot и boo, но не be или b;

[ ] (символы в квадратных скобках) соответствуют любому (одному) символу, записанному в квадратных скобках. Например, [blv]it соответствует bit, lit, vit;

[^] ("крышечка" в квадратных скобках) в начале строки шаблона внутри [ ] означает отрицание. Например, [^blv] соответствует любым символам, за исключением b, l и v;

[-] (черточка в пределах [ ]) задает диапазон символов. Например, [a-o] соответствует любому символу от a до o;

\ (обратная косая черта) перед символом шаблона заставляет рассматривать его "буквально". Например, \^ соответствует символу ^, а не задает шаблон.

Команда Replace ("горячая" клавиша Ctrl-Q A) не только находит заданную строку, но и заменяет ее другой. Окно диалога команды содержит дополнительное поле ввода замещающей строки и несколько новых опций. Большинство элементов окна диалога Replace совпадает с рассмотренными ранее элементами окна Find .

Дополнительное поле Prompt on replace контролирует необходимость приглашения подтвердить замену найденного текста. Новая кнопка управления Change All автоматически выполняет операцию замены для

всех найденных строк в тексте. Кнопка OK заменяет только первую найденную строку.

Команда меню Search - Go to Line Number требует ввода номера строки, на которую будет установлен курсор.

3.1.6. Задание опций интегрированной среды

Прежде чем начать работу в IDE, следует проверить и при необходимости выставить нужные опции. Далее предельно кратко рассматриваются только некоторые из них.

Меню Options включает команды Compiler, Transfer..., Make..., Linker..., Application..., Debugger..., Directories.... Все опции имеют значения по умолчанию, "впечатанные" в программу интегрированной среды и изменяемые с помощью программы настройки IDE (BCINST.EXE). Многие из опций дублируются в файле конфигурации среды (стандартное имя - TCCONFIG.TC). Для того чтобы начать работать в IDE, прежде всего требуется задать директории, используемые текстовым редактором, компилятором и компоновщиком. Для этого используется команда Directories, и включает:

1.Окно ввода Include Directories используется для задания директориев заголовочных файлов. В окне разрешается указывать несколько директориев, разделяемых символом ';'.

2. Окно ввода Library Directories задает директории, содержащие объектный файл загрузчика (C0S.0BJ) и файлы библиотек функций (LIB). Окно ввода Output Directory задает директорий, в который помещаются файлы с расширениями .OBJ, .EXE и .MAP. Если в окне - пустая строка, используется текущий директорий.

При выборе строки Compiler меню Options открывается еще одно меню для настройки опций компилятора. Меню включает команды Code generation..., Entry/Exit Code..., C++ options..., Optimizations..., Source..., Messages..., Names.... Наиболее важные для дальнейшего обсуждения опции задаются при выборе команды Code generation через окно диалога.

Опция считается выбранной, если она помечена символом (•), и включенной, если она помечена символом [x].

Более "тонкие" опции доступны для установки при нажатии кнопки управления More.... На экране появляется окно диалога.

Используя это окно, можно установить необходимый режим генерации кода для арифметики с плавающей точкой, задать набор машинных команд, которые будет использовать компилятор. Важными для дальнейшего изложения являются две "отладочные" опции: Line numbers debug info и Debug info in OBJs. Когда эти опции включены, они значительно увеличивают размер генерируемого кода, но при этом облегчают отладку программ с использованием символического отладчика TD.EXE.

Опции оптимизации, редактирования связей, генерации ошибок и предупреждений позволяют управлять самыми тонкими моментами в работе интегрированной среды.

3.1.7. Компиляция, редактирование связей, запуск программы на выполнение

Запуск программы на компиляцию выполняется либо через команду Compile to OBJ меню Compile, либо нажатием "горячей" клавиши Alt - F9. Команда Make EXE file меню Compile также запускает программу на компиляцию и при отсутствии синтаксических ошибок автоматически запускает компоновщик для получения EXE-файла. Этой команде соответствует "горячая" клавиша F9. Еще одна возможность для запуска программы на компиляцию - команда Run меню Run ("горячая" клавиша Ctrl-F9). После успешной компиляции и компоновки Run запускает полученный EXE-файл на выполнение.

Все сообщения об ошибках и предупреждения IDE помещает в окно по имени Messages. Это окно активно после завершения компиляции. Если в программе обнаружены ошибки или предупреждения, включаются средства трассировки ошибки, которые связывают строки текста программы в окне редактора со строками окна Message. Перемещение высвечивания клавишами со стрелками в окне Message синхронно сопровождается высвечиванием соответствующих ошибочных строк в тексте программы. При нажатии клавиши ENTER активизируется окно

редактора и курсор устанавливается на ошибочную строку. Повторное нажатие клавиши F6 вновь делает активным окно Message.

3.1.8. Многофайловая компиляция

При работе с большими программами намного удобнее размещать части программы не в одном, а в нескольких файлах. Каждый файл должен включать целиком одну или несколько функции. Затем через специальный файл - файл проекта - интегрированная среда узнает, какие из текстовых файлов следует объединить в исполняемый (.EXE) файл.

Работа IDE Borland C++ с файлами проектов удобна и проста. Все необходимые для этого команды (Open project..., Close project, Add item..., Delete item. Local options.... Include files...) включены в меню Project

Прежде всего файл проекта должен быть открыт. Для этого выполняется команда Project Open- Project, открывается окно диалога, позволяющее загрузить нужный файл или создать новый файл с заданным именем. Работа с окном диалога подобна работе с окном Open, рассмотренной ранее. Интегрированная среда использует имя файла проекта в качестве имен .EXE и .MAP-файлов. Отметим, что файлы проектов в IDE Borland C++ содержат все опции, открытые окна и их положение на экране (desktop). При открытии уже существующего файла проекта сохраненные опции и экран восстанавливаются.

После нажатия клавиши ENTER или кнопки ОК среда активизирует окно с заголовком Project

Одна из строк окна Project высвечивается. Если создан новый файл проекта, окно первоначально будет пустым. Включение файлов в проект и их удаление выполняются либо через команды Add item... и Delete item... меню Project, либо нажатием клавиш Ins и Del. При добавлении файлов в проект открывается окно диалога, позволяющее выбрать нужный файл.

Нужное имя файла высвечивается в окне Files. После этого нажимается либо клавиша ENTER, либо кнопка Add. Выбранное имя файла добавляется в окно Project сразу же после высвеченной строки и вновь высвечивается окно диалога. Так продолжается до тех пор, пока не будет нажата кнопка действий Done.

Окно Project упрощает переход от одного файла, включенного в проект, к другому при их редактировании. Для этого высвечивание перемещается на нужную строку окна Project и нажимается либо клавиша ENTER, либо два раза левая кнопка "мыши".

3.2.Отладка программ

3.2.1 Общие положения

Как правило, большинство смысловых ошибок в программе можно обнаружить, используя встроенный в интегрированную среду отладчик. С его помощью можно проследить ход исполнения программы, текущие значения переменных и внутренних регистров процессора и при необходимости установить нужные значения. Отыскать наиболее замысловатые ошибки, отладить резидентные программы и драйверы позволяет специальный внешний отладчик -- программа TD.EXE (здесь и далее турбо отладчик).

3.2.2 Отладчик интегрированной среды Borland C++

Для того чтобы отладка программы в IDE стала возможной, необходимо выполнить компиляцию и компоновку программы с включенной опцией Source Debugging, расположенной в меню опций отладчика (Main Menu-Options-Debugger).

Встроенный отладчик позволяет:

1) выполнить программу по шагам, строка исходного текста за строкой;

2) выполнить программу до заданной строки, называемой далее точкой останова(Breakpoint);

3) проследить изменение заданных переменных программы и при необходимости установить для них новые значения.

Команда Trace into из меню Run ("горячая клавиша F7") запускает программу на отладку. Интегрированная среда высвечивает строку программы, содержащую точку входа main(). После этого нажатием клавиши F7 вызывают выполнение кода, соответствующего одной строке текста программы на Си. Если в строке записана ссылка на функцию,

начинается трассировка по тексту функции. (Библиотечные функции выполняются без трассировки за одно нажатие клавиши F7). При необходимости выполнения строки Си-текста за один шаг используется клавиша F8 (команда меню Run Step over).

Для ускорения процесса отладки используется команда Go to cursor меню Run ("горячая" клавиша F4). Программа выполняется до строки, в которой в данный момент располагается текстовый курсор. Например, если программа попадает в цикл и не хочется выполнять по шагам все его итерации, необходимо переместить курсор на следующую после тела цикла строку текста и нажать клавишу F4. Точно также можно пропустить ту часть отлаживаемой программы, детальный анализ которой не требуется.

Другая возможность ускоренного выполнения отлаживаемой программы -использование точек останова. Включение точки выполняет команда Toggle breakpoint меню Debug ("горячая "клавиша Ctrl-F8). Точка помещается в той строке программы, где располагается текстовый курсор. Повторное выполнение команды Toggle breakpoint удаляет точку останова. Строка Си-текста с точкой останова высвечивается на экране другим цветом.

Когда исполняющая программа достигает точки останова, IDE проверяет условие, которое может быть задано для неё. Если условие не задано, эта точка считается точкой безусловного останова (Unconditional Breakpoint). Здесь выполнение программы всегда останавливается и может быть продолжено либо по шагам (клавиша F7), либо до курсора (клавиша F4), либо до следующей точки останова (клавиша Ctrl-F9).

Если же с точкой останова связано какое-то условие, она называется условной (Conditional Breakpoint). Выполнение программы здесь останавливается только, если в этот момент вычисленное условие истинно.

Задание условий выполняется через окно диалога, открываемое при выполнении команды Breakpoints... меню Debug. Нажатие кнопки управления Delete удаляет точку останова из текста программы. Поле Pass указывает число прохождений точки до останова.

Если необходимо задать или скорректировать условие останова, используется окно диалога, открываемое при нажатии кнопки управления Edit.

Исключительно полезное свойство встроенного отладчика IDE - возможность наблюдения за изменением значений переменных в ходе выполнения программы. Интегрированная среда использует для этого специальное окно с именем Watch. Оно появляется сразу же после нажатия клавиши F7. Задание имён переменных или выражений, называемых далее точками наблюдения(Watches), выполняется командой Watches меню Debug. Её выполнение открывает подменю, содержащее команды управления точками наблюдения: Add watch..., Delete watch, Edit watch..., Remove all watches.

Команда Add Watch ("горячая" клавиша Ctrl-F7 в случае, когда активно текстовое окно) добавляет в окно Watch новую точку наблюдения. При выполнении команды открывается окно диалога, с использованием которого задаётся любое допустимое выражение языка Си, в том числе имя переменной, и через запятую специфицируется формат представления переменной. По умолчанию выражением является слово в текущей позиции курсора активного окна редактирования, а формат выбирается по типу переменной.

После того, как завершён ввод выражения, нажимается клавиша ENTER и в окне Watch появляется новая строка, показывающая текущее значение вычисленного выражения. В таблице приведены спецификаторы формата, распознаваемые IDE в отладочных выражениях.

Таблица. Спецификаторы формата для отладочных выражений

Символ формата	Производимое действие
c	Отображает значение как ASCII-символ.
d	Отображает целое число в десятичной системе счисления.
f[n]	Отображает число с плавающей точкой; n-небязательный спецификатор число знаков после десятичной точки.
h или x	Отображает целое число в шестнадцатеричной системе счисления.

m	Отображает коды байтов (дамп) области памяти, выделенной под переменную.
p	Отображает указатель в форме segment:offset. Для near-указателей segment содержит имя сегментного регистра(DS, ES, CS, SS)
r	Отображает значение каждого поля структуры или объединения.
s	Отображает значение как ASCII-строку.

Окно watch, когда оно активно, позволяет выполнить добавление, удаление и редактирование точки наблюдения без использования команды подменю Watch. Высвеченная в окне Watch строка соответствует текущей точке наблюдения. Ее редактирование осуществляется простым нажатием клавиши ENTER. В окно диалога будет копироваться текущее выражение. Для удаления текущей точки наблюдения нажимается клавиша Del. Добавление новой точки наблюдения происходит при нажатии клавиши Ins. Команда Remove all watches подменю Debug-Watches полностью очищается окноWatch.

В ходе отладки программы можно сразу вносить изменения в её текст. После каждого изменения отладчик запрашивает подтверждение на продолжение сеанса отладки или предлагает повторную компиляцию программы. Если необходимо начать выполнение программы сначала, не дожидаясь её завершения, используется клавиша Ctrl-F2.

Завершая обсуждение встроенного отладчика IDE, отметим, что в большинстве случаев его возможностей достаточно для поиска и устранения практически всех типов смысловых ошибок. Однако существует несколько случаев, когда необходимо использовать внешний турбо-отладчик TD.EXE. Первый из них из них связан с использованием регистровых псевдопеременных. Этот эффективный, но рискованный приём профессионального программирования всегда требует особой осторожности из-за возможности неожиданного изменения значений регистров. Второй случай – это тщательная проработка программных деталей при оптимизации производительности программы и её размера. Третий случай – это отладка резидентно завершающихся программ. И,

наконец, четвёртый случай имеет место, когда причина ошибки неясна даже после тщательного исследования поведения программы средствами встроенного отладчика.

3.3. Структура и компоненты простой программы

Индивидуальное задание

Создать файл проекта, разработать, откомпилировать и отладить Си-программу.

1. Даны два действительных числа a и b. Получить их сумму, разность и произведение.

2. Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.

3. Даны два действительных положительных числа. Найти среднее арифметическое и среднее геометрическое этих чисел.

4. Даны два действительных числа. Найти среднее арифметическое этих чисел и среднее геометрическое из модулей.

5. Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.

6. Дана сторона равностороннего треугольника. Найти площадь этого треугольника.

7. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

8. Найти площадь кольца, внутренний радиус которого равен a (вводится с клавиатуры), а внешний—заданному числу r (r > a).

9. Найти сумму членов арифметической прогрессии

a, a+d, ..., a+(n—1)d

по данным значениям a, d, n.

10. Вычислить расстояние между двумя точками с координатами (x1, y1) и (x2, y2).

11. Треугольник задан координатами своих вершин. Найти периметр треугольника.

12. Дано действительное число x. Не пользуясь никакими другими арифметическими операциями, кроме умножения, сложения и вычитания, вычислить

$2x^4-3x^3+4x^2-5x+6$

Разрешается использовать не более четырех умножений и четырех сложений и вычитании.

13. Дано действительное число x. Не пользуясь никакими другими арифметическими операциями, кроме умножения, сложения и вычитания, вычислить

$1- 2x+3x^2-4x^3$ и $1+ 2x+3x^2+4x^3$

Разрешается использовать не более восьми операций.

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема : программирование ветвящихся структур.

Цель : изучить условный оператор, оператор выбора, условную тернарную операцию, научиться программировать разветвляющиеся алгоритмы.

Краткие теоретические сведения

Операторы ветвления выбирают в программе из группы альтернатив возможное продолжение вычислительного процесса. Выбор выполняется исходя из значения заданного выражения. В Си используются два оператора ветвления: if... else и switch, а также условная операция.

4.1. Условный оператор

Оператор if имеет следующую общую форму записи:

```
if (cond_expression) TRUE_statement
   [else FALSE_stateinent]
```

При выполнении оператора if сначала вычисляется выражение-условие cond_expression. Если результат - ИСТИНА (любое отличное от нуля значение), выполняется оператор TRUE_statement. Если результат выражения - ЛОЖЬ (равен 0), то выполняется оператор FALSE_stateinent. Если ключевое слово else отсутствует, то в этом случае оператор TRUE_statement пропускается, а управление передается на следующий после if оператор.

Операторы TRUE_statement и FALSE_statement сами могут быть операторами if, образуя так называемые вложенные if. Компилятор интерпретирует вложенные if, сопоставляя каждое из ключевых слов else с последним встретившимся словом if, не имеющим " своего" else. Соответствие ищется в пределах блока, в который заключено слово if. Внутренние и внешние блоки при этом не рассматриваются. Если соответствия для if не найдено, компилятор полагает, что if не имеет ветви else. Например,

```
int a, b;
if(b > 0)
   a = 1;
else
   if(b==0)
       a=0;
   else
       a=-1;
```

то же самое, но используя два оператора if:

```
int a, b;
if(b > 0)
    a=1;
if(b==0)
    a=0;
else
    a=-1;
```

В результате переменной a будет присваиваться значение
-1, если b < 0,
+1, если b > 0, и
0, если b = 0.

Операция условия ? является частным случаем оператора if... else. Например, фрагмент

```
if (cond_expression)
   x =y;
else
   x= y+4;
```

эквивалентен более короткой записи:

```
x=  (cond_expression) ?  y: y + 4;
```

4.2. Организация мультиветвления. Операторы switch и break.

Часто возникающая в программировании задача - выбор одного варианта из многих. Можно это сделать с помощью вложенных if ... else. Однако более удобный способ - использование оператора switch, общий формат которого таков:

```
switch (switch_expression)
{ case constant1: statement1; [break;]
  case constanti: statementi; [break;]
  case constantN: statementN; [break;]
  [default:       statementN+1; ]
}
```

Оператор switch выполняется так. Сначала вычисляется значение выражения switch_expression. Тип значения должен быть одним из целых. Вычисленное значение сравнивается со значениями констант или константных выражений $constant_1$, ..., $constant_N$. При совпадении значения switch_expression с $constant_i$ выполняется оператор $statement_i$. Затем управление передается на оператор сразу после switch, если в i-й ветви присутствует оператор break (оператор прерывания процесса). В противном случае выполняются операторы в ветвях i+1, i+2 и так далее до тех пор, пока в них не встретится оператор break или не будет выполнен оператор $statement_{N+1}$.

Если значение switch_expression не совпало ни с одной из констант $constant_1$,..., $constant_N$, выполняется оператор в ветви, помеченной default. При ее отсутствии выполняется следующий после конструкции switch оператор.

Индивидуальное задание

Составить схему алгоритма, создать файл проекта, разработать, откомпилировать и отладить Си-программу.

1. Осуществить проверку того, что три пары чисел, представляющих декартовые координаты точек на плоскости, являются вершинами некоторого равнобедренного треугольника. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

2. Осуществить проверку того, что три пары чисел, представляющих декартовые координаты точек на плоскости, являются вершинами некоторого равностороннего треугольника. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

3. Осуществить проверку того, что три пары чисел, представляющих декартовые координаты точек на плоскости, являются вершинами некоторого прямоугольного треугольника. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

4. Заданы три числа, представляющих собой длины отрезков. Определить можно ли построить треугольник с такими длинами сторон и, если это возможно – определить тип треугольника: равносторонний, равнобедренный, прямоугольный, иной треугольник.

5. Определить, пересекаются ли два отрезка на плоскости, заданные декартовыми координатами своих концов. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

6. Даны действительные числа a, b,c. Выяснить, имеет ли уравнение $ax^2+bx+c=0$ действительные корни. Если действительные корни имеются, то найти их. В противном случае ответом должно служить сообщение, что действительных корней нет. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

7. Даны действительные числа a, b, c ($a\neq0$). Полностью исследовать биквадратное уравнение $ax^4 + bx^2+c=0$, т. е. если действительных корней нет, то должно быть выдано сообщение об этом, иначе должны быть выданы два или четыре корня.

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема : программирование циклических структур.

Цель : изучить способы организации циклических процессов: циклов со счетчиком, предусловием и постусловием.

Краткие теоретические сведения

5.1. Организация циклов со счетчиком

Для организации циклов со счетчиком используется оператор for. Он имеет следующий формат:

```
for ( выражение 1 ; выражение 2 ; выражение 3 )
    тело цикла
```

Выражение 1 обычно используется для установления начального значения переменных, управляющих циклом.

Выражение 2 - это выражение, определяющее условие, при котором тело цикла будет выполняться.

Выражение 3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла.

Схема выполнения оператора for:

1. Вычисляется выражение 1.

2. Вычисляется выражение 2.

3. Если значения выражения 2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение 3 и осуществляется переход к пункту 2, если выражение 2 равно нулю (ложь), то управление передается на оператор, следующий за оператором for.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным.

Пример:

```
   void main(){
     int i,b;

     for (i=1; i<10; i++)
         b=i*i;
     return;
}
```

В этом примере вычисляются квадраты чисел от 1 до 9.

Некоторые варианты использования оператора for повышают его гибкость за счет возможности использования нескольких переменных, управляющих циклом.

Пример:

```
 void main(){
   int top, bot;
   char string[100],temp;

  for   (top=0,bot=100;   top<bot;   top++,bot--){
    temp=string[top];
    string[bot]=temp;
   }
   return;
 }
```

В этом примере, реализующем запись строки символов в обратном порядке, для управления циклом используются две переменные top и bot. Отметим, что на месте выражение 1 и выражение 3 здесь используются несколько выражений, записанных через запятую (операция последовательного вычисления), и выполняемых последовательно.

Другим вариантом использования оператора for является бесконечный цикл. Для организации такого цикла можно использовать пустое условное выражение, а для выхода из цикла обычно используют дополнительное условие и оператор break.

Пример:

```
for (;;){
   ...
   ...
   break;
   ...
}
```

5.2. Организация циклов с предусловием

Оператор цикла while называется циклом с предусловием и имеет следующий формат:

```
while (выражение)
    тело цикла;
```

В качестве выражения допускается использовать любое выражение языка Си, а в качестве тела любой оператор, в том числе пустой или составной. Схема выполнения оператора while следующая:

1. Вычисляется выражение.
2. Если выражение ложно, то выполнение оператора while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполняется тело оператора while.
3. Процесс повторяется с пункта 1.

Оператор цикла вида

```
for (выражение-1; выражение-2; выражение-3)
    тело цикла;
```

может быть заменен оператором while следующим образом:

```
выражение-1;
while (выражение-2){
  тело цикла;
  выражение-3;
}
```

Так же как и при выполнении оператора for, в операторе while вначале происходит проверка условия. Поэтому оператор while удобно

использовать в ситуациях, когда тело оператора не всегда нужно выполнять.

5.3. Организация циклов с постусловием

Оператор цикла do while называется оператором цикла с постусловием и используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз. Формат оператора имеет следующий вид:

```
do
    тело цикла;
while (выражение);
```

Схема выполнения оператора do while :

1. Выполняется тело цикла (которое может быть составным оператором).
2. Вычисляется выражение.
3. Если выражение ложно, то выполнение оператора do while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполнение оператора продолжается с пункта 1.

Чтобы прервать выполнение цикла до того, как условие станет ложным, можно использовать оператор break.

Важной особенностью циклов с постусловием является то, что тело в цикле do while выполняется по крайней мере один раз.

Операторы циклов for, while и do while могут быть вложенными.

5.4. Операторы передачи управления

5.4.1. Оператор break

Оператор break обеспечивает прекращение выполнения самого внутреннего из объединяющих его операторов switch, do, for, while. После выполнения оператора break управление передается оператору, следующему за прерванным.

5.4.2. Оператор continue

Оператор continue, как и оператор break, используется только внутри операторов цикла, но в отличие от него выполнение программы продолжается не с оператора, следующего за прерванным оператором, а с начала прерванного оператора.

Пример:

```
void main(){
  int a,b;

  for (a=1,b=0; a<100; b+=a,a++){
    if (b%2) continue;
       ...    /* обработка четных сумм  */
    }
  return 0;
}
```

Когда сумма чисел от 1 до a становится нечетной, оператор continue передает управление на очередную итерацию цикла for, не выполняя операторы обработки четных сумм.

Оператор continue, как и оператор break, прерывает самый внутренний из объемлющих его циклов.

Индивидуальное задание

Составить схему алгоритма, создать файл проекта, разработать, откомпилировать и отладить Си-программу.

1. Определить является ли произвольное положительное целое число простым. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

2. Даны натуральные числа n, m. Найти наибольший общий делитель двух заданных натуральных чисел. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

3. Даны натуральные числа n, m. Найти наименьшее общее кратное двух заданных натуральных чисел. Программа должна обеспечивать ввод

исходных данных, вывод результатов и диагностических сообщений в особых случаях.

4. Составить программу, выполняющую вывод на экран всех простых чисел, меньше заданного. Необходимо предусмотреть обработку ошибок ввода и вывод диагностических сообщений для всех исключительных ситуаций.

5. Определить, являются ли два произвольных натуральных числа взаимно простыми. Необходимо обеспечить ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

6. Составить программу, выполняющую разложение произвольного положительного целого числа на элементарные множители. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

7. Проверить, является ли произвольное целое число целой положительной степенью простого числа. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

8. Напечатать таблицу истинности для логической функции F=(A и B) или ¯(B или C) в следующем виде:

A	B	C	F
True	true	true	true
True	true	false	true
True	false	true	false
…….	……	……..	……
False	false	false	true

9. Напечатать в возрастающем порядке все трёхзначные числа, в десятичной записи которых нет одинаковых цифр (операции деления не использовать).

10. Числа Фибоначчи (f_n) определяются формулами

$f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$ при n=2,3, ...

Необходимо определить число Фибоначчи с номером i.

11. Напечатать таблицу значений функции sin x на отрезке [a,b] (параметры a и b целые числа) с шагом 0.1 в виде десяти столбцов.

Считать, что точность результата составляет четыре значащих десятичных разряда.

x	0.0	0.1	. . .	0.9
0	0.0000	0.0998	. . .	0.7833
1	0.8415	0.8912	. . .	0.9463

12. Напечатать таблицу значений функции cos x на отрезке [a,b] (параметры a и b целые числа) с шагом 0.1 в виде десяти столбцов. Считать, что точность результата составляет четыре значащих десятичных разряда. Пример таблицы приведен в задаче 11.

ЛАБОРАТОРНАЯ РАБОТА № 6

Тема : разработка пользовательских подпрограмм.

Цель: изучить особенности программирования Си-функций.

Краткие теоретические сведения

5.1. Определение и вызов функций

Мощность языка СИ во многом определяется легкостью и гибкостью в определении и использовании функций в СИ-программах. В отличие от других языков программирования высокого уровня в языке СИ нет деления на процедуры, подпрограммы и функции, здесь вся программа строится только из функций.

Функция - это совокупность объявлений и операторов, обычно предназначенная для решения определенной задачи. Каждая функция должна иметь имя, которое используется для ее объявления, определения и вызова. В любой программе на СИ должна быть функция с именем **main** (главная функция), именно с этой функции, в каком бы месте программы она не находилась, начинается выполнение программы.

При вызове функции ей при помощи аргументов (формальных параметров) могут быть переданы некоторые значения (фактические параметры), используемые во время выполнения функции. Функция может возвращать некоторое (одно!) значение. Это возвращаемое значение и есть результат выполнения функции, который при выполнении программы подставляется в точку вызова функции, где бы этот вызов ни встретился. Допускается также использовать функции не имеющие аргументов и функции не возвращающие никаких значений. Действие таких функций может состоять, например, в изменении значений некоторых переменных, выводе на печать некоторых текстов и т.п.

С использованием функций в языке СИ связаны три понятия - определение функции (описание действий, выполняемых функцией), объявление функции (задание формы обращения к функции) и вызов функции.

Определение функции задает тип возвращаемого значения, имя функции, типы и число формальных параметров, а также объявления переменных и операторы, называемые телом функции, и определяющие действие функции.

Пример:

```
int rus (unsigned char ch){
   if (ch>='А' && ch<='Я')
       return 1;
   else
       return 0;
}
```

В данном примере определена функция с именем rus, имеющая один параметр с именем ch и типом unsigned char. Функция возвращает целое значение, равное 1, если параметр функции является буквой русского алфавита, или 0 в противном случае.

В языке СИ нет требования, чтобы определение функции обязательно предшествовало ее вызову. Определения используемых функций могут следовать за определением функции main, перед ним, или находится в другом файле.

Однако для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров до вызова функции нужно поместить объявление (прототип) функции.

Объявление функции имеет такой же вид, что и определение функции, с той лишь разницей, что тело функции отсутствует, и имена формальных параметров тоже могут быть опущены. Для функции, определенной в последнем примере, прототип может иметь вид

int rus (unsigned char ch); или **rus (unsigned char);**

В программах на языке СИ широко используются, так называемые, библиотечные функции, т.е. функции предварительно разработанные и записанные в библиотеки. Прототипы библиотечных функций находятся в специальных заголовочных файлах, поставляемых вместе с библиотеками в составе систем программирования, и включаются в программу с помощью директивы #include.

Рекомендуется всегда задавать прототип функции. Это позволит компилятору либо выдавать диагностические сообщения, при неправильном использовании функции, либо корректным образом регулировать несоответствие аргументов устанавливаемое при выполнении программы.

В соответствии с синтаксисом языка СИ определение функции имеет следующую форму:

[спецификатор-типа] имя-функции ([список-формальных-параметров])
{ тело-функции }

Спецификатор-типа функции задает тип возвращаемого значения и может задавать любой тип. Если спецификатор-типа не задан, то предполагается, что функция возвращает значение типа **int**.

Функция не может возвращать массив или функцию, но может возвращать указатель на любой тип, в том числе и на массив и на функцию. Тип возвращаемого значения, задаваемый в определении функции, должен соответствовать типу в объявлении этой функции.

Функция возвращает значение если ее выполнение заканчивается оператором return, содержащим некоторое выражение. Указанное выражение вычисляется, преобразуется, если необходимо, к типу возвращаемого значения и возвращается в точку вызова функции в качестве результата. Если оператор return не содержит выражения, или выполнение функции завершается после выполнения последнего ее оператора (без выполнения оператора return), то возвращаемое значение не определено. Для функций, не использующих возвращаемое значение, должен быть использован тип **void**, указывающий на отсутствие возвращаемого значения. Если функция определена как функция, возвращающая некоторое значение, а в операторе return при выходе из нее отсутствует выражение, то поведение вызывающей функции после передачи ей управления может быть непредсказуемым.

Список-формальных-параметров - это последовательность объявлений формальных параметров, разделенная запятыми. Формальные параметры - это переменные, используемые внутри тела функции и получающие значение при вызове функции путем копирования в них значений соответствующих фактических параметров. Список-формальных-

параметров может заканчиваться запятой (,) или запятой с многоточием (,...), это означает, что число аргументов функции переменно. Однако предполагается, что функция имеет, по крайней мере, столько обязательных аргументов, сколько формальных параметров задано перед последней запятой в списке параметров. Такой функции может быть передано большее число аргументов, но над дополнительными аргументами не проводится контроль типов.

Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово void.

Порядок и типы формальных параметров должны быть одинаковыми в определении функции и во всех ее объявлениях. Типы фактических параметров при вызове функции должны быть совместимы с типами соответствующих формальных параметров. Тип формального параметра может быть любым основным типом, структурой, объединением, перечислением, указателем или массивом. Если тип формального параметра не указан, то этому параметру присваивается тип int.

Тело функции - это составной оператор, содержащий операторы, определяющие действие функции.

Все переменные, объявленные в теле функции без указания класса памяти, имеют класс памяти auto, т.е. они являются локальными. При вызове функции локальным переменным отводится память в стеке и производится их инициализация. Управление передается первому оператору тела функции и начинается выполнение функции, которое продолжается до тех пор, пока не встретится оператор return или последний оператор тела функции. Управление при этом возвращается в точку, следующую за точкой вызова, а локальные переменные становятся недоступными. При новом вызове функции для локальных переменных память распределяется вновь, и поэтому старые значения локальных переменных теряются.

Параметры функции передаются по значению и могут рассматриваться как локальные переменные, для которых выделяется память при вызове функции и производится инициализация значениями фактических параметров. При выходе из функции значения этих

переменных теряются. Поскольку передача параметров происходит по значению, в теле функции нельзя изменить значения переменных в вызывающей функции, являющихся фактическими параметрами. Однако, если в качестве параметра передать указатель на некоторую переменную, то используя операцию разадресации можно изменить значение этой переменной.

Пример:

```
void change (int *x, int *y){
  int k=*x;
  *x=*y;
  *y=k;
}
```

При вызове такой функции в качестве фактических параметров должны быть использованы не значения переменных, а их адреса

change (&a,&b);

Если требуется вызвать функцию до ее определения в рассматриваемом файле, или определение функции находится в другом исходном файле, то вызов функции следует предварять объявлением этой функции. Объявление (прототип) функции имеет следующий формат:

[спецификатор-типа] имя-функции ([список-формальных-параметров]) [,список-имен-функций];

В отличие от определения функции, в прототипе за заголовком сразу же следует точка с запятой, а тело функции отсутствует. Если несколько разных функций возвращают значения одинакового типа и имеют одинаковые списки формальных параметров, то эти функции можно объявить в одном прототипе, указав имя одной из функций в качестве имени-функции, а все другие поместить в список-имен-функций, причем каждая функция должна сопровождаться списком формальных параметров. Правила использования остальных элементов формата такие же, как при определении функции. Имена формальных параметров при объявлении функции можно не указывать, а если они указаны, то их область действия распространяется только до конца объявления.

Прототип - это явное объявление функции, которое предшествует определению функции. Тип возвращаемого значения при объявлении

функции должен соответствовать типу возвращаемого значения в определении функции.

Наличие в прототипе полного списка типов аргументов параметров позволяет выполнить проверку соответствия типов фактических параметров при вызове функции типам формальных параметров, и, если необходимо, выполнить соответствующие преобразования.

В прототипе можно указать, что число параметров функции переменно, или что функция не имеет параметров.

Вызов функции имеет следующий формат:

адресное-выражение ([список-выражений])

Поскольку синтаксически имя функции является адресом начала тела функции, в качестве обращения к функции может быть использовано адресное-выражение (в том числе и имя функции или разадресация указателя на функцию), имеющее значение адреса функции.

Список-выражений представляет собой список фактических параметров, передаваемых в функцию. Этот список может быть и пустым, но наличие круглых скобок обязательно.

Фактический параметр может быть величиной любого основного типа, структурой, объединением, перечислением или указателем на объект любого типа. Массив и функция не могут быть использованы в качестве фактических параметров, но можно использовать указатели на эти объекты.

Выполнение вызова функции происходит следующим образом:

1. Вычисляются выражения в списке выражений и подвергаются обычным арифметическим преобразованиям. Затем, если известен прототип функции, тип полученного фактического аргумента сравнивается с типом соответствующего формального параметра. Если они не совпадают, то либо производится преобразование типов, либо формируется сообщение об ошибке. Число выражений в списке выражений должно совпадать с числом формальных параметров, если только функция не имеет переменного числа параметров. В последнем случае проверке подлежат только обязательные параметры. Если в прототипе функции указано, что ей не требуются параметры, а при вызове они указаны, формируется сообщение об ошибке.

2. Происходит присваивание значений фактических параметров соответствующим формальным параметрам.

3. Управление передается на первый оператор функции.

4. Выполнение оператора return в теле функции возвращает управление и возможно, значение в вызывающую функцию. При отсутствии оператора return управление возвращается после выполнения последнего оператора тела функции, а возвращаемое значение не определено.

Любая функция в программе на языке СИ может быть вызвана рекурсивно, т.е. она может вызывать саму себя. Компилятор допускает любое число рекурсивных вызовов. При каждом вызове для формальных параметров и переменных с классом памяти auto и register выделяется новая область памяти, так что их значения из предыдущих вызовов не теряются, но в каждый момент времени доступны только значения текущего вызова.

Переменные, объявленные с классом памяти static, не требуют выделения новой области памяти при каждом рекурсивном вызове функции и их значения доступны в течение всего времени выполнения программы.

Классический пример рекурсии - это математическое определение факториала n! :

n! = 1 при n=0;

n*(n-1)! при n>1 .

Функция, вычисляющая факториал, будет иметь следующий вид:

```
long fakt(int n)
{
 return ( (n==1) ? 1 : n*fakt(n-1)  );
}
```

Хотя компилятор языка СИ не ограничивает число рекурсивных вызовов функций, это число ограничивается ресурсом памяти компьютера и при слишком большом числе рекурсивных вызовов может произойти переполнение стека.

5.2. Передача параметров функции main

Функция main, с которой начинается выполнение СИ-программы, может быть определена с параметрами, которые передаются из внешнего окружения, например, из командной строки. Во внешнем окружении действуют свои правила представления данных, а точнее, все данные представляются в виде строк символов. Для передачи этих строк в функцию main используются два параметра, первый параметр служит для передачи числа передаваемых строк, второй для передачи самих строк. Общепринятые (но не обязательные) имена этих параметров argc и argv. Параметр argc имеет тип int, его значение формируется из анализа командной строки и равно количеству слов в командной строке, включая и имя вызываемой программы (под словом понимается любой текст не содержащий символа пробел). Параметр argv это массив указателей на строки, каждая из которых содержит одно слово из командной строки. Если слово должно содержать символ пробел, то при записи его в командную строку оно должно быть заключено в кавычки.

Функция main может иметь и третий параметр, который принято называть argp, и который служит для передачи в функцию main параметров операционной системы (среды) в которой выполняется СИ-программа.

Заголовок функции main имеет вид:

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

Если, например, командная строка СИ-программы имеет вид:

```
A:\>cprog working 'C program' 1
```

то аргументы argc, argv, argp представляются в памяти как показано в схеме на рис.5.1.

```
argc  [ 4  ]
argv  [    ]--> [    ]-->   [A:\cprog.exe\0]
                  [    ]-->   [working\0]
                  [    ]-->   [C program\0]
                  [    ]-->   [1\0]
                  [NULL]
```

Рис.5.1. Схема размещения параметров командной строки

Операционная система поддерживает передачу значений для параметров argc, argv, argp, а на пользователе лежит ответственность за передачу и использование фактических аргументов функции main.

Следующий пример представляет программу печати фактических аргументов, передаваемых в функцию main из операционной системы и параметров операционной системы.

Пример:

```
int main ( int argc, char *argv[], char *argp[]){
  int i=0;
  printf ("\n Имя программы %s", argv[0]);
  for    (i=1; i>=argc; i++)
     printf ("\n аргумент %d равен %s", argv[i]);
  printf ("\n Параметры операционной системы:");
  while  (*argp){
    printf ("\n %s",*argp);
    argp++;
  }
  return (0);
}
```

Индивидуальное задание

1. Заданы два комплексных числа (a+ib) и (c+id) и тип операции (умножение, сложение, вычитание, деление). Определить функции выполнения арифметических операций над комплексными числами и вычислить значения модулей заданных комплексных чисел.

2. Разработать функцию, результатом работы которой является отнесение символа к одному из следующих типов: цифра, латинский символ, русский символ, и вывод шестнадцатеричного кода символа.

4. Числа Фибоначчи (fn) определяются формулами

f0=f1=1; fn=fn-1+fn-2 при n=2,3, ...

Необходимо разработать функцию печати n-первых чисел последовательности Фибоначчи.

5. Два простых числа называются “близнецами”, если они отличаются друг от друга на 2 (таковы, например, числа 41 и 43). Напечатать все пары “близнецов” из отрезка [n,2n], где n - заданное целое число, больше 2.

6. Два натуральных числа называются “дружественными”, если каждое из них равно сумме всех делителей другого, кроме самого этого числа (таковы, например, числа 220 (110,55,44,22,20,11,10,5,4,2,1) и 284 (142,71,4,2,1)). Напечатать все пары “дружественных” чисел, не превосходящих заданного натурального числа.