Министерство образования Республики Беларусь
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра экономической информатики

МЕТОДИЧЕСКОЕ ПОСОБИЕ И УЧЕБНЫЕ МАТЕРИАЛЫ

по курсу

«Основы информатики и вычислительной техники»

для студентов экономических специальностей

заочной формы обучения

В 2-х частях

Часть 1

МИНСК 2000

УДК 002.5 (075)
ББК 32.81 Я 73
М54

Авторы: А.В.Бахирев, Е.Н. Живицкая, В.Н. Комличенко, С.А. Соколов.

Методическое пособие и учебные материалы по курсу М54 «Основы информатики и вычислительной техники» для студентов экономических специальностей заочной формы обучения. В 2ч. Ч.1. А.В.Бахирев, Е.Н. Живицкая, В.Н. Комличенко и др. -Мн.: БГУИР, 2000.- с.82: Ил.12. ISBN 985-444-077-X (ч.1)

В работе представлены: основные темы лекционного курса «Основы информатики и вычислительной техники»; методические рекомендации по выполнению контрольных работ; пример программной реализации типового задания, входящего в состав контрольной работы; список используемой литературы и варианты контрольных работ.

УДК 002.5 (075)
ББК 32.81 Я 73

ISBN 985-444-077-X (ч.1)
ISBN 985-444-078-8

Предисловие

Пособие содержит справочные материалы по читаемому курсу лекций, а также основам программирования на языке С, самостоятельное изучение которых может оказаться затруднительным. Цель - максимально просто и постепенно ввести пользователя в этот сложный раздел информатики. Так как дисциплины информатики очень динамичны и программа ежегодно пересматривается, ее текст в пособии не приводится . Программа курса и перечень вариантов контрольных заданий должна предоставляться студентам заочникам дополнительно на установочной сессии в электронном или печатном виде. Пособие содержит также ряд приложений, организующих и направляющих работу обучаемого.

Приложение 1 - описывает этапы разработки, кодирования и отладки программы, рассмотренной на примере решения типовой задачи по учету персонала фирмы. Аналогичные задачи предлагаются в составе заданий контрольных работы. Алгоритмы разрабатываемых функций должны быть описаны и представлены в виде блок-схем в пункте «Описание программы» пояснительной записки контрольной работы.

Приложение 2 - содержит краткое описание настройки и основных режимов работы в интегрированной среде Borland C++, версии 2.0 или выше, рекомендуемой для выполния заданий по программированию. Рассматривается содержание основных этапов создания программ в интегрированной среде. Крайне желательно освоить работу с отладчиком. Это позволит Вам отслеживать процесс выполнения программы и реальное изменение состояния ее переменных. Нельзя заявить, что Вы освоили основы программирования, если Вы не можете работать с отладчиком.

Материалы по многофайловой компиляции не являются необходимыми для решения задач в контрольных работах. Это раздел для студентов, желающих в более полном объеме постичь «секреты» разработки программ.

Приложение 3 - не является необходимыми для программирования задач в контрольных работах. Это информация для «продвинутых» студентов - тех, кто разобрался с материалами по многофайловой компиляции и овладел приемами составления программных многофайловых проектов. Приложение содержит информацию об организации межфайловых связей (между отдельно компилируемыми файлами) в программе посредством использования внешних переменных.

Приложение 4 содержит требования по содержанию пояснительной записки, представляющей результаты проделанной работы. Пункты 1 – 8 определяют рекомендуемое (возможное) содержание пояснительной записки.

Приложение 5 содержит перечень требование к оформлению пояснительной записки, основанных на требованиях ГОСТ, предъявляемых к оформлению научных работ и ЕСПД (единой системы программной документации).

Приложение 6 содержит образец оформления титульного листа пояснительной записки.

1.Введение в предметную область

1.1.Информатика и информация

Информатика – это отрасль науки, изучающая информационные процессы в различных областях знаний. Предметом информатики является выявление и изучение свойств информации, а также вопросов, связанных с ее сбором, хранением, поиском, переработкой, преобразованием, распространением и использованием в различных сферах деятельности.

Термином **информация** обычно обозначают обмен сведениями между людьми (в общем случае между источником и приемником), которые расширяют понимание объекта или явления. Родоначальник теории информации Клод Шеннон определил: “Информация – это снятая неопределенность”.

Различают содержательную и формальную структуру информации. Содержательная структура ориентированна на содержание информации (научные знания, гипотезы, теории, законы). Формальная структура ориентирована на представление информации (символьная, текстовая, графическая, звуковая). Информация – ресурс, объем которого с течением времени лавинообразно возрастает. В настоящее время объем информации ежегодно удваивается.

Информация стала товаром первой необходимости, но ее все возрастающий объем не позволяет воспринять и сориентироваться в ней должным образом.

Следует различать понятия **данные** и **знания**.

Данные – это информация, представленная для обработки в удобном виде. Данные, как правило, хранят в каком-либо упорядоченном виде: в массивах и списках.

Знания - это проверенный практикой результат познания действительности, ее верное отражение в сознании человека.

Экономическая информация – словосочетание, введенное в обиход с внедрением средств вычислительной техники в управление хозяйственной деятельностью. Под **экономической информацией** понимают информацию о процессах общественного производства, обмена и потребления материальных благ. По назначению в процессе управления экономическая информация подразделяется на управляющую (командную) и осведомляющую (например учетно-статистическую).

Экономическая информация:

- *специфична по форме представления*, отражается в виде первичных и связных, юридически оформленных (т.е. имеющих подписи) на традиционных электронных документах;
- *объемна*, т.е. содержит детальную информацию о процессах, для управления которыми она используется;

- *циклична*, т.к. для большинства производственных и хозяйственных процессов характерна повторяемость составляющих их стадий и соответствующей информации, описывающей эти процессы;
- *специфична по способам обработки* (преобладают арифметические и логические операции, а результаты представляются в виде текстов, таблиц, диаграмм, графиков).

Наиболее важными требованиями, предъявляемыми к экономической информации, являются:

- корректность, т.е. однозначность для всех потребителей;
- ценность - относительное (для разных потребителей - разная) свойство, проявляющееся в том случае, если информация используется для достижения цели;
- оперативность, отражает актуальность информации для необходимых расчетов и принятия решений в изменяющихся условиях;
- точность - определяет допустимый уровень искажений входной, выходной и др. типов информации;
- достоверность - свойство информации отражать реально существующие объекты с необходимой точностью;
- устойчивость - способность реагировать на изменение исходных данных без нарушения необходимой точности;
- достаточность - означает, что она содержит минимально необходимый объем сведений, необходимых для принятия правильного решения.

Исследования экономической информации позволили:

- классифицировать информацию по месту возникновения (входящая, исходящая); по участию в процессе обработки/хранения (исходная, производная, хранимая без обработки, промежуточная, результирующая); по отношению к функциям управления (плановая, прогнозная, нормативная, конструкторско-технологическая, учетная, финансовая и т.д.);
- выявить ряд особенностей, влияющих на организацию автоматизированной обработки.

Информатика как наука возникла в результате целенаправленного изучения информационных процессов.

В последнее время под информатикой понимают науку об описании, осмыслении, интерпретации, представлении, формализации и применении знаний с помощью средств вычислительной техники для поиска нового знания во всех областях деятельности человека.

Экономическая информатика (ЭИ) выявляет и изучает свойства экономической информации, закономерности ее переработки; процессов управления переработкой информации в искусственных, социальных и биологических системах и направлена на решение экономических задач. Ее важнейшими назначениями являются: сбор, преобразование и регистрация информации, обработка и хранение, преобразование, тиражирование, распространение и использование для решения экономических задач по управлению экономическими объектами.

1.2.Информация и управление

Определяющей областью в процессе деятельности человека является организационное управление, понимаемое как процесс целенаправленного воздействия на объект, организующий его функционирование по некоторой заданной программе.

С точки зрения информационных процессов **управление** состоит из следующих составляющих:

- выработка управляющим органом управляющей информации, соответствующей цели (программе) управления;
- передача управляющей информации объекту управления;
- получение и анализ реакции объекта;
- корректировка или выработка новой управляющей информации с целью оптимизации функционирования объекта управления.

Система управления экономическим объектом представляет собой человеко-машинный комплекс, в основе которого определяются следующие подсистемы:

- информационное обеспечение - это система классификации и кодирования информации, технологическая схема обработки данных, нормативно-справочная информация документооборота;
- организационное обеспечение - совокупность мер и мероприятий, регламентирующих функционирование системы управления, ее описание, инструкции и регламенты обслуживающему персоналу;
- техническое обеспечение - комплекс используемых в системе управления технических средств, в том числе ЭВМ и средств связи;
- математическое обеспечение - совокупность методов, правил, математических моделей и алгоритмов решения задач;
- лингвистическое обеспечение - совокупность терминов и искусственных языков, правил формации естественного языка;
- программное обеспечение - совокупность программ, систем обработки данных и документов, необходимых для эксплуатации этих программ;
- правовое обеспечение - совокупность правовых норм, определяющих создание, юридический статус и функционирование системы.

Управление экономическим объектом - основная часть информационной технологии решения экономической задачи. Ее важнейшими процедурами являются: сбор, регистрация, хранение, обработка, преобразование, тиражирование, распространение и использование информации. В процессе управления объектом экономическая информация подвергается, как правило, всем процедурам, в ряде случаев часть из них может отсутствовать. Последовательность процедур также может быть различной. Некоторые из них могут повторяться. Их состав и особенности зависят от экономического

объекта, ведущего автоматизированную обработку информации, и процессов, протекающих в среде его обитания.

1.3.Информационные технологии

Технология (от греч. Techne –искусство, мастерство, умение) есть совокупность методов обработки, изготовления, изменения состояния, свойств, формы сырья, материала или полуфабриката, осуществляемых в процессе производства. Под **информационной технологией** (ИТ) обычно понимают совокупность методов, способов, приемов и средств обработки документированной информации, включая прикладные программные средства и регламентированный порядок следования их применений, а также совокупность всех видов информационной техники. ИТ ориентируются на получение, обработку и распространение информации.

Становление и развитие технологий стремительно изменяет окружающий мир. С самого начала эти процессы рассматривались в тесной связи с экономическими объектами, т.к. процесс всякой деятельности осуществляется по технологии, определяемой целью, предметом, средствами, характером операции и результатами.

Понятие «технологический процесс» в экономической литературе фактически было вытеснено термином «**бизнес-процесс**». Эти понятия во многом совпадают.

В основе описания бизнес-процессов лежат понятия:

- объект - информационный, материальный или финансовый, используемый в бизнес-процессе (оборудование, счет);
- событие – внешнее (не контролируемое в рамках процесса) действие, произошедшее с объектом (получение письма, поломка оборудования, начисление штрафа);
- операция – элементарное действие, выполняемое в рамках рассматриваемого бизнес-процесса (посылка письма, оплата счета);
- исполнитель – должностное лицо, ответственное за выполнение одной или нескольких операций бизнес-процесса (менеджер, сотрудник архива, директор).

Жизненный цикл объекта связан с внешними событиями и операциями, выполняемыми исполнителями в составе ПРОЦЕССА.

ИТ играют важную роль в поддержке бизнес-процесса. Они проникли во все виды деятельности человека, т.к. позволяют интегрировать различные виды технологий, синтезировать и накапливать информацию для внедрения в практику в соответствии с общественными потребностями. Целью широкого распространения ИТ является решение проблемы информатизации общества, понимаемой как распространение и внедрение комплекса мер, направленных на своевременное использование достоверной информации во всех сферах человеческой деятельности.

Информатизация общества – повсеместное внедрение комплекса, направленного на обеспечение полного и своевременного использования достоверной информации обобщенных знаний во всех социально значимых видах человеческой деятельности.

Информатизация – это реакция общества на существенный рост информационных ресурсов и потребность в значительном увеличении производительности труда в информационном секторе общественного производства, где сосредоточено около половины (в США более 60 %, в СНГ около 40 %) трудоспособного населения.

Считается, что внедрение ИТ повысит результативность решений, принимаемых на всех уровнях управления. Это обеспечит как рост экономических показателей развития хозяйства страны, так и получение качественных научных достижений в функциональных и прикладных науках, направленных на развитие производства, создание новых рабочих мест, повышение жизненного уровня населения, т.е. улучшение «качества жизни».

Темпы оснащения вычислительной техникой всех аспектов человеческой деятельности, развитие, совершенствование и усложнение компьютерных технологий остро ставят вопрос об уровне подготовки кадров, работающих в области эффективного применения компьютерных информационных систем. Для подготовки таких кадров недостаточно бытового понимания компьютерных технологий и поверхностных знаний о компьютере и используемых программных системах. Цель данной работы – компенсировать недостаточное обеспечение студентов учебной литературой и оказать помощь в освоении вопросов, которые представляют сложность при самостоятельном изучении.

2.Основные сведения об архитектуре и работе компьютера

2.1.Основные блоки компьютера

Любой компьютер состоит из 4 основных частей:

- устройства ввода информации;
- устройства обработки информации;
- устройства хранения информации;
- устройства вывода информации;

Конструктивно эти части могут объединяться в одном корпусе (в компьютерах класса Notebook – записная книжка) или же каждая может состоять из нескольких достаточно громоздких устройств (в больших компьютерах, аналогично ЕС ЭВМ). С конструктивной точки зрения в компьютере можно выделить следующие основные устройства:

- системный блок;
- видеомонитор;
- клавиатура;
- принтер;
- мышь.

Для решения ряда специальных задач компьютер может быть оснащен звуковыми колонками, сканером для ввода графических изображений с помощью специальных

цветных фломастеров, дигитайзеров для автоматизированного ввода в компьютер чертежей, схем, географических карт.

Для хранения больших объемов информации на специальной магнитной ленте ПК может быть укомплектован стриммером.

В современных мощных ПК в качестве устройства ввода информации используются устройства CD-ROM, считывающие ее с компакт-дисков. Такие накопители позволяют работать с большими (650 Мб) объемами информации, но не позволяют изменить ее.

Системный блок компьютера – сердце компьютера. Внутри системного блока компьютера располагаются:

- блок питания;
- большая системная (называется материнская) плата, в которую вставляются платы поменьше - контроллеры;
- жесткий диск или винчестер (hard disk);
- 1 или 2 дисковода (disk drive или floppy drive) для работы с дискетами (floppy disk);
- дисковод для компакт-дисков (CD-ROM drive);
- небольшой громкоговоритель;
- много соединительных кабелей.

Материнская плата является сердцем любой компьютерной системы. Чаще всего она представляет собой плоский лист фольгированного стеклотекстолита, покрытого зеленым лаком. Схему соединений платы нужной конфигурации получают травлением медной фольги. Используются двухслойные и многослойные платы. В материнскую плату вставляются платы поменьше - контроллеры. Устройства ввода-вывода (мышь, клавиатура, видеомонитор, принтер) подключаются либо непосредственно к материнской плате, либо к контроллерам.

На материнской плате имеется большая микросхема – центральный процессор, выполняющий обработку данных и управление компьютером. Важнейшим устройством также являются микросхемы оперативной памяти оперативно запоминающего устройства (ОЗУ), характеризующиеся малым временем доступа к информации.

Микропроцессор – это кусочек кремния, выращенный в стерильных условиях. Он представляет собой совокупность простых транзисторов, соединенных определенным образом на кремниевой пластине, которая называется интегральной схемой, или ИС , или чипом. Большинство микропроцессоров имеют специально встроенные области памяти, называемые регистрами, в которых они осуществляют все манипуляции и расчеты. Сигналы, перемещающиеся по микропроцессору, представляют собой серию цифровых импульсов. Их перемещение происходит практически параллельно. Каждая серия импульсов представляет собой отдельную команду микропроцессора.Каждая команда имеет идентифицирующее имя. Полный набор реализующих команд и соответствующих функций называют множеством микрокоманд процессора.

Внутренняя структура кремния определяет алгоритм программы работы микропроцессора для каждого входного сигнала. Эта программа называется микрокодом микропроцессора.

Помимо выполнения внутреннего программного кода микропроцессор должен получать (входную) и выдывать (выходную) информацию. Для реализации этой связи используется так называемая микропроцессорная шина данных.

Для работы с данными внешней среды микропроцессору необходимо знать, где хранятся данные во внешней среде. Для этой цели используется еще одна шина, называемая адресной.

Микропроцессоры различаются по имеющимся в их наличии ресурсам, что влияет на скорость их работы. Они могут отличаться как числом, так и размерами самих регистров. Размеры определяются числом бит, с которыми он может работать одновременно, т.е. 16-битному микропроцессору необходим один или более регистров размерностью в 16 бит. **Бит**- это двоичный разряд, принимающий значение 0 или 1 и являющийся наименьшим количеством информации. Наряду с битом широко используются следующие понятия: байт, равный 8 бит; килобайт(Кб) =1024 байт, мегабайт (Мб)=1024 Кб, гигабайт (Гб)=1024 Мб.Байт информации используется для кодирования и представления в ПЭВМ одного символа (буквы, цифры, знака и т.п.)
На скорость обработки информации оказывает влияние и число бит в шине данных. Микропроцессоры с 8-,16-,32- битными шинами данных используются различными модификациями компьютеров IBM.
Число бит адресной шины влияет на объем адресной памяти, например, микропроцессор с 16-адресными линиями может работать с 65536 различными ячейками памяти (64 К).

История развития микропроцессора прошла путь от 4-битного Intel Corporation (1971 г.) микропроцессора 4004 до современных 32-битовых .В компьютерах, начиная с I 80386, началась новая эра в программировании. Была снята сегментация памяти (64к), связанная с 16 битами адресной шины. Объем адресуемой памяти вырос на 4 Гб, появилась возможность устранить ограничения, накладываемые первоначальными конструкциями микропроцессоров и соответствующими операционными системами.

Кроме того, современные микропроцессоры снабжены сверхоперативной кеш-памятью. Эта специальная встроенная память позволяет загружать в нее код нескольких следующих команд кода программного обеспечения, прежде чем в этом коде появится необходимость, что помогает микропроцессору работать без задержек, связанных с загрузкой очередной команды из оперативной памяти. На IBM совместимых компьютерах, как правило, используются микропроцессоры фирмы INTEL
I 4004 – 1971 г. – калькуляторы (4 бит)
I 8080 – 1974 г. - 8 бит
I 8086 – 1976 г. - 16 бит, с возможностью обработки 1 Мб памяти, разделенного на 16....64 Кб сегментов

I 8088 – модификация (I 8086) 16-битного упрощения 8-битной шиной данных
I 80286 – 20-битная адресная шина с частотой до 20Мгц. и возможностью работать с 1Гб памяти, 16 Мб физической и 1008 Мб виртуальной.
I 80386 – 32-битная с тактовой частотой от 16 Мгц и выше (до 32) с возможностью адресации до 42 Гб памяти.

Работа микропроцессора. Микропроцессор реагирует на каждый конкретный входной сигнал одним и тем же определенным образом. Последовательность бит, поступающая в микропроцессор приказывает ему выполнить определенную операцию. Например, сложение требует около 7 инструкций. Каждая конкретная команда говорит микропроцессору, где конкретно брать числа,что приводит к небольшим вариациям.

Числа , с которыми работает микропроцессор, должны быть размещены в одном из 3-х мест: в регистрах микропроцессора, в оперативной памяти (RAM), либо в самой микрокоманде. Числа из внешней памяти должны быть сначала считаны в ОП.
Команды микропроцессора заносят числа в его регистры, обрабатывают их, а затем записывают результат в память или во входной порт устройства.

Микропроцессоры могут выполнять только такие простые пошаговые инструкции в двоичной системе исчисления.
В процессе программирования происходит замена множества стандартных пошанговых операций одной командой.

Сопроцессор – специальная интегральная схема, которая работает совместно с главным процессором. Сопроцессор – обычный микропроцессор, не столь универсальный, как главный. Он настраивается на выполнение определенной специфической функции, например, математической операции или графического представления и выполняет ее во много раз быстрее, чем главный процессор. Его деятельность определяется главным процессором. Множества команд процессора и сопроцессора не совпадают. Программы для сопроцессора пишутся специальным образом, поэтому сам по себе сопроцессор не улучшает производительности компьютера. Если программа не использует микрокоманд сопроцессора, то скорость ее выполнения не увеличится при наличии сопроцессора.

2.2.Память компьютера

Память компьютера делится на 2 типа: основную и вспомогательную.
Основная – это память, к которой микропроцессор может непосредственно обратиться. Такую память называют **оперативной** (ОЗУ), потому что процессор может обратиться к ней в любой момент. Доступ к памяти получается через адресную шину, либо через порт ввода-вывода. Так как доступ реализуется к любому байту ОЗУ, то такая память называется памятью с прямым доступом – Random Access Memory (RAM). Память, требующая более значительных временных затрат, называется **внешней**. Она может в десятки и сотни раз превышать по объему внутреннюю память. Однако, чтобы данные

из внешней памяти были обработаны микропроцессором, они должны быть занесены из внешней памяти в ОЗУ.

Хранение информации. Работа с памятью основывается на простой концепции. Память должна быть способна сохранить бит информации так,чтобы он мог быть потом извлечен оттуда. **Динамическая память –** устройство, базирующееся на способности сохранять электрический заряд – конденсаторы. Конденсатор способен сохранять заряд (свое состояние) несколько микросекунд -–время, за которое специальные схемы обеспечивают его подзарядку – обновление информации. Память, реализованная на таких принципах, называется динамической.

В современных компьютерах конденсаторы заменены специальными цепями проводников. Большое их количество объединяется в одном корпусе. Однако, как и в обычных конденсаторах, информация такой микросхемы должна постоянно обновляться.

Статическая память. В отличие от динамической (удерживающей заряд) памяти статическая память позволяет потоку электронов циркулировать по цепи. Существуют только два направления движения. Это позволяет использовать данные цепи в качестве элементов памяти. Статическая память работает подобно механическому выключателю, имеющему два положения. Переключатель, управляемый электрическим током, известен как реле. Память первых компьютеров создавалась на основе электрических реле.

Транзисторы, удовлетворяющие всем требованиям переключателя, объединенные в единую цепь, исполняют роль чипов статической памяти. Статическая, как и динамическая память, нуждается в постоянном источнике питания, чтобы поддерживать ее состояние.

Постоянная память. Наиболее важная информация компьютера, как правило, не должна изменяться во времени. Для такой информации предназначена постоянная память, получившая название ROM, – память (Read Only Memory) или постоянное запоминающее устройство (ПЗУ). Компьютер не может изменять информацию в ПЗУ, а только читать ее. Такие микросхемы изготавливаются по специальной технологии, которая не только создает нужные цепи на керамическом кристалле, но и заносит в него нужную информацию.

Альтернативой непрограммируемых ПЗУ являются программируемые ПЗУ. Такой тип микросхемы содержит массивы элементов, программирование которых может выполняться после их изготовления специальным ПЗУ – программатором (ПЗУ – прожигается). Такой процесс называют прожиганием ПЗУ (технология взрыва элемента). Такие ПЗУ обеспечивают только одно программирование и не могут корректироваться. Альтернативные стираемые ПЗУ (Erasable Programmable Read – Only Memory (EPROM)) позволяют стирать хранящуюся информацию и заносить в них новую при помощи ультрафиолетового луча, через специальное прозрачное окошко на корпусе.

Архитектура памяти. Память в PC имела достаточно простую структуру. Она была представлена одним блоком, в котором каждый байт был доступен по указанию его

адреса. Микросхемы памяти разбивались на девять банков. Восемь микросхем выделяют по одному биту для организации каждого байта памяти, т.е. в каждом байте присутствовало по одному биту из различных микросхем с одним и тем же адресом. Девятая микросхема использовалась в качестве контрольного бита четкости. Этот бит всегда выставлялся таким образом, чтобы общее число бит в байте было нечетным.

Время обращения. Время обращения зависит не только от скорости работы микропроцессора, но и от скорости работы микросхем памяти, которые должны согласовываться. Так как обычно скорость работы микропроцессора превышает скорость работы микросхем памяти, для их согласования используют ввод необходимых, например в динамической памяти для обновления информации, циклов ожидания, а также специальные решения по архитектуре памяти и использованию более быстродействующих микросхем.

Для комплектации современных компьютеров, на примере 80386 процессора, используются микросхемы с временем доступа 120, 80 наносекунд и более скоростных. Важной характеристикой микросхемы является кроме времени доступа, время цикла, которое говорит о том, как быстро можно произвести повторное обращение. Более производительными, но и более дорогими, являются статические микросхемы, т.к. для них нет необходимости в обновлении информации.

Альтернативой является кэш – память (быстрая память), в которую заносится последующий программный код. Это помогает обойтись или уменьшить количество циклов ожидания процессора. Практически используемые объемы такой памяти 16-256Кб. Обычно работа с кэш-памятью управляется отдельным контроллером.

Разбивка оперативной памяти на страницы и замена части ее быстрыми специальными микросхемами, работа с которыми осуществляется как с кэш-памятью, также позволяет значительно уменьшить количество циклов ожидания, возникающих теперь только при переходе с одной страницы на другую. При этом существенную роль имеет размер страниц, т.к. чем больше страница, тем больше вероятность того, что следующий бит будет внутри текущей страницы (обычно размер страниц 2 или 4 Кб). Страничная организация выполняется при помощи специальных микросхем ОЗУ статического режима.

Одной из эффективных архитектур организаций памяти является и разделение памяти на два или более банков. Последовательность бит при этом хранится в разных банках при чтении информации, поэтому процессор обращается то к одному, то к другому банку. Во время обращения к одному банку, другой реализует цикл обновления информации, и поэтому микропроцессору не придется ждать.

Логическая организация памяти. При разработке PC конструкторы разделили всю память на разделы и каждый раздел предназначили для реализации специфических функций. Результирующую диаграмму называют картой памяти.

Память в пределах до 640 Кб называют базовой памятью.

В ее нижних адресах располагалась зона векторов прерываний и зона данных BIOS (0-128) и (1024-1280), зона программ DOS до 640Кб. Программы, написанные для MS DOS, могут использовать только эту память. Следующие 128 Кб были зарезервированы для буфера видеоадаптера.

Последние 256 Кб памяти использовались как адресное пространство ПЗУ, в которых мог, например, располагаться интерпретатор БЕЙСИКа либо другие программы.

Дополнительная память. Память, выходящая за пределы 1 Мбайта, обычно называют дополнительной памятью, или extended-памятью. Эта память доступна в защищенных режимах Intel 80286, I80386 процессорах. Для MS DOS, работающей в реальном режиме , такая память не доступна, но очень часто используется, например, для организации виртуального диска.

Одним из методов преодоления 640-Кб ограничения была организация расширенной Expended Memory Specification, или EMS-памяти. Работа с этой памятью основывалась на переключении при помощи специального устройства банков памяти по 16 Кбайт из дополнительной области размером 8 Мбайт в нормальное адресное пространство центрального микропроцессора. Для этого использовалась зона карты памяти в диапазоне от 784 до 960 Кбайт и специальные платы расширения с переключателями банков на них.

Дальнейшее развитие EMS стандарта дало принципиальную возможность реализовать многозадачный режим.

2.3.Многозадачный режим работы

Реальная работа в многозадачном режиме была реализована в I80286 процессоре, который мог переключаться для работы в реальном (обычном режиме 8086) или защищенном.

В защищенном режиме память рассматривается как разделенная на сегменты, а специальный механизм адресации и защиты позволяет поддерживать работу в защищенном режиме. Защита памяти позволила организовать среду таким образом, чтобы в ней могли параллельно выполняться несколько программ – каждая в своей области защищенной памяти. Дальнейшее развитие защищенного режима и появление нового виртуального было реализовано в 80386 - процессоре фирмы INTEL. Этот процессор потребовал и наличие новой операционной системы, которая сделала бы доступными новые его возможности для прикладных программ.

Понятие сегмента памяти сохранило свой первоначальный смысл, однако теперь сегмент мог иметь размер до 4 Гбайт. Программы могли считать, что память реализована как линейная, операционная система получила возможность переключать процессор по своему усмотрению в режим 8086, 80286 на работу с 32-разрядными приложениями, написанными для 386 процессора.

Следует различать работу 386 процессора в виртуальном режиме и 8086 процессора, в котором 386 процессор используется главным образом как более быстрый процессор от того же термина в отношении памяти (виртуальной памяти).

Виртуальная память. Понятие виртуальной памяти используется для определения совокупности реальной физической оперативной памяти и свободной памяти на жестком диске.

Для работы с виртуальной памятью необходимо специальное программное обеспечение. В качестве операционной системы для 386 процессора была разработана система Windows95. Виртуальная память системы Windows состоит из ОП компьютера и файла подкачки, расположенного на жестком диске. Операционная система управляет объемом доступной программы памяти, перекачивая программные сегменты и сегменты данных из памяти на жесткий диск и обратно.

Если сегменты не находятся в ОП, операционная система помечает его отсутствие. В случае если необходимо выполнить оператор из данного сегмента, 386 процессор генерирует состояние “сегмент отсутствует”, сообщая операционной системе, что доступ к данному сегменту невозможен. ОС загрузит данный сегмент в доступную область ОП и передаст управление программе, которая вызвала соответствующее прерывание.

Для увеличения производительности компьютера 386 процессор и следующие за ним модели процессоров позволяют реализовать страничную организацию памяти. Объем используемых страниц 4 Кбайта памяти. При этом сегмент памяти может содержать одну или более таких страниц.

2.3.1.Система Windows 95

Первоначально система Windows использовалась как графический расширитель интерфейса MS-DOS. Совершенствуясь от версии к версии, она приобрела функции законченной операционной системы (ОС), которой, однако, присуща совместимость с MS-DOS.

В Windows 95 (рис. 2.1) операционная среда, поддерживающая работу всех Windows приложений, представляет собой Системную виртуальную машину, способную обслуживать как 32-разрядные, так и 16-разрядные приложения, используемые в версии 3.1.

32- разрядные приложения – это новые приложения Windows, использующие 32-разрядную модель памяти процессоров, начиная с 80386, и подмножество интерфейса программирования Win 32. Каждое из этих приложений в Windows 95 имеет своё собственное адресное пространство, не доступное для других приложений.

Оболочка Windows 95 – это 32-разрядное приложение, которое обеспечивает взаимодействие пользователя и системы. Она объединяет функции диспетчера программ, диспетчера файлов и диспетчера задач в одном приложении.

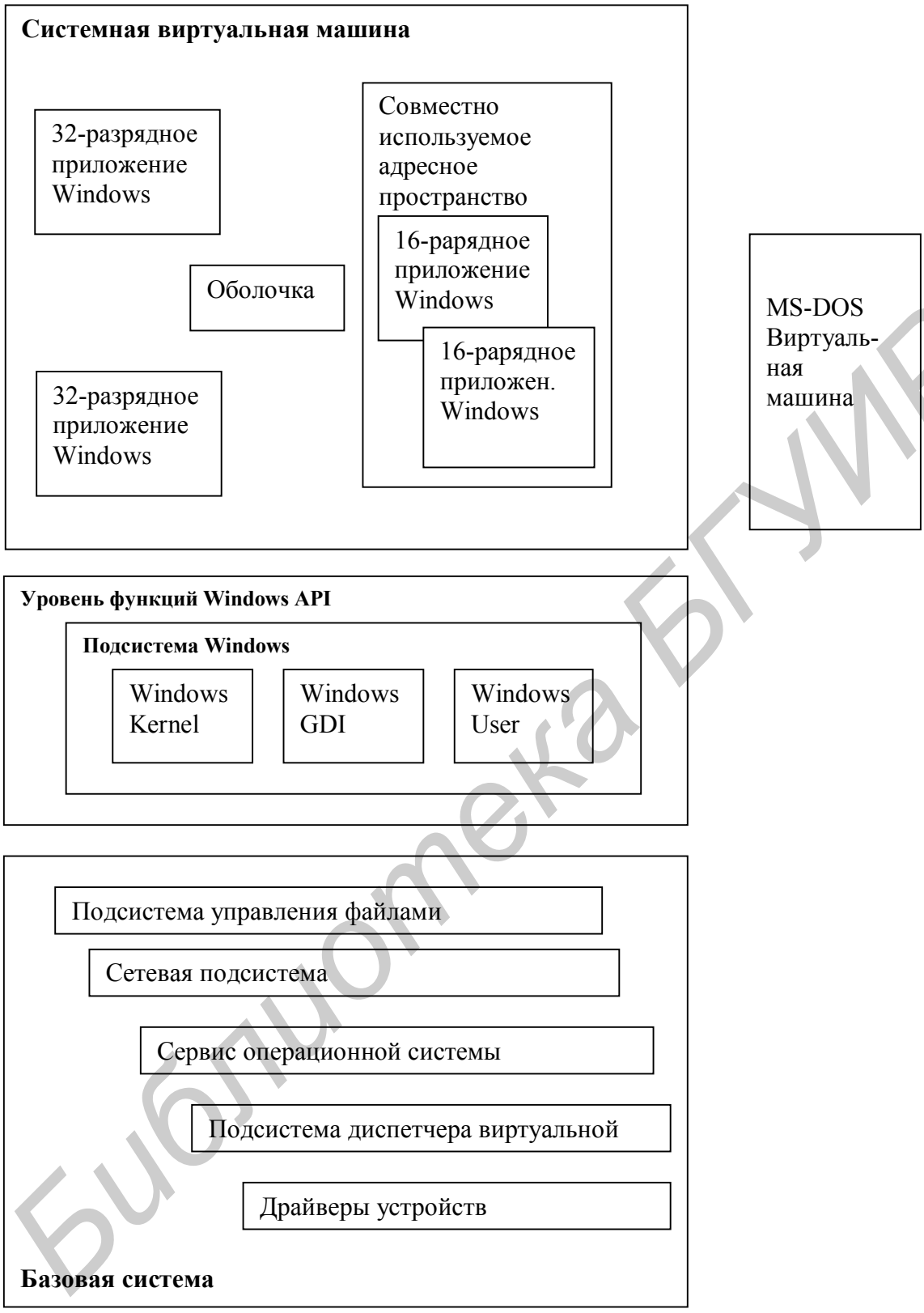


Рис.2.1. Архитектура системы Windows 95

Обслуживание многозадачности реализуется на основе принципа так называемой вытесняющей многозадачности.

2.3.2.Вытесняющая многозадачность

Этот термин означает, что система Windows 95 сама (в зависимости от внутренней ситуации) передает управление тому или иному приложению либо отбирает его от него для передачи другому, в отличие от кооперативной многозадачности, которая предусматривалась в Windows 3.1/3.11. При кооперативной многозадачности выполняемые приложения периодически проверяли так называемую очередь сообщений для того, чтобы при возможности передать управление другому приложению. При этом приложения, редко проверяющие очередь сообщений, забирали на себя львиную долю ресурсов системы, и пока они трудились, другие приложения простаивали. Для 16-разрядных приложений Windows 95 по-прежнему предусматривает кооперативную многозадачность, хотя и лучше использует свои ресурсы, чем Windows 3.1/3.11.

Windows 95 имеет эффективные средства управления потоками и реализует многопоточную работу. Такая работа дает более быструю реакцию на действие пользователя, может происходить в фоновом режиме. Потоки в свою очередь могут порождать новые потоки. Это свойство используют приложения, написанные под Windows 95, обеспечивая проведение одновременно нескольких сложных операций. Например, работая с текстовым редактором, можно одновременно работать с письмом к другу, выполнять длительную проверку орфографии текста отчета, форматировать текст большой книги и печатать подготовленную ранее статью для журнала.

16-разрядные приложения используют модель сегментной адресации памяти (20-битную адресацию), по сути модель памяти процессора 80286. 16-разрядные приложения при работе под Windows 95, как и в Windows 3.1, делят между собой единое адресное пространство и не могут управляться в соответствии с принципом вытесняющей многозадачности.

Низкоуровневые функции Windows, такие, как динамическое выделение памяти, обеспечивает модуль Windows Kernel как для 32-разрядных, так и для 16-разрядных приложений.

Графические возможности Windows обеспечивает модуль GDI.

Для создания и управления на экране окнами, кнопками, панелями и другими видимыми элементами интерфейса предназначен модуль USER.

Для обеспечения работы MS-DOS приложений запускаются MS-DOS виртуальные машины.

2.3.3.Базовая система

Остальные модули Windows 95 реализуют функции базовой системы.

К ним относятся:

- модули управления файловой системой;

- модули сетевой подсистемы, отвечающие за поддержку локальной сети в Windows 95;
- подсистема конфигурирования аппаратных средств;
- диспетчер виртуальной машины (управление задачами, загрузкой, завершением, памятью и взаимодействием программ);
- драйверы устройств, в том числе драйверы реального времени, а также виртуальные драйверы, обеспечивающие совместное использование устройств (например, запуск двух виртуальных машин MS-DOS, для каждой из которых требуется вывод на единый физический экран).

2.3.4.Программные приложения

1. Программы для работы с текстами:

- Текстовые редакторы (текстовые процессоры), издательские системы (программы верстки) - это программы для набора, редактирования и подготовки к печати любых документов от нескольких слов на одной странице до цветного иллюстрированного журнала или многотомного издания. Наиболее распространенными являются: Лексикон,Microsoft Word,Word for Windows,Write,AmiPro,WordPerfect,Aldus Page Maker, Xerox Ventura Publisher,Quark Xpress.
- Программы прверки правописания (спеллеры)-SpellRus, Корректор, Арф, Орфо, Пропись, Глагол.
- Программы-переводчики - с английского, немецкого и других языков на русский и обратный перевод –Socrat,Stylus.
- Программы – словари - Контекст, Русский филолог, Lingvo.
- Программы распознавания образов - CuneiForm,FineRider.

2. Графические редакторы:

- Растровые (битмэповые) - рисуют изображение по точкам - для каждой точки задается цвет- AldusPhotoStyler,Adobe PhotoShop,PhotoFinish,Picture Man,Paintbrush.
- Векторные - рисуют линиями (отрезками прямой, дугами) – Corel Draw, Adobe Illustrator, MS Draw.
- Программы поддержки сканера –Desk Skan ll.
- Программы компьютерной анимации – AutoDesk Animator, Alias Power Animator, 3D Studio,The Animation Stend, Picture Man.
- Программы для обработки видеоизображений – Media 100, D/Vision-Pro.

3. Базы данных, бухгалтерские программы, электронные таблицы:

- Системы управления базами данных (СУБД) – dBase, FoxPro, Paradox, Clipper, Clarion, MS Access.
- Бухгалтерские программы, программы для ведения офисной документации, программы планирования финансовой, коммерческой и производственной деятельности – 1С: Бухгалтерия, Гепард, Босс, Офис, Турбо-бухгалтер, Финансы без проблем, TimeLine.
- Электронные таблицы (spreadsheet) – Microsoft Excel, Lotus 1-2-3, Quattro Pro.

4. Музыкальные редакторы:

- Программы-секвенсеры предназначены для обработки музыкальных файлов – Cubase, CakeWalk, Midisoft Recording Session, Encore.
- Программы для обработки звуковых файлов – SoundForge, Wave, Cool, Dart, Sound Recorder.

5. Интегрированные пакеты:

Представляют собой комплекс полностью совместимых между собой программ, обеспечивающий пользователю единую в своей основе комфортную среду – Microsoft Office, Lotus SmartSuite, MS Worcs.

6. Телекоммутационные и сетевые программы:

- Программное обеспечение пользователя (Client Software) для совместного использования ресурсов.
- Программное обеспечение на сервере(Server Software) для объединения и разделения ресурсов.
- Сетевые протоколы - правила коммуникаций.

7. Оболочки для работы с языками программирования. Предназначены для удобства написания и отладки программ- GW Basic, Quic Basic, Visual Basic, Borland Pascal, Turbo Pascal, Borland C++,Turbo C++, Visual C++, Symantec C++.

8. Математические программы. Используются для построения графиков, решения уравнений и систем уравнений – MathLab, MatCad.

9. Профессиональные программные комплексы для компьютерного конструирования. Предназначены для конструирования как отдельных узлов и деталей , так и технологических линий – AutoCAD.

10. Развивающие и обучающие программы. Представлены семейством обучающих, контролирующих, обучающе-контролирующих, тестирующих, развивающих игровых программ, программ-тренажеров и т.п.. Ориентированы в большинстве случаев на самостоятельную подготовку учащихся, проведения и поддержку лабораторных занятий. Особенно эффективны для хорошо формализуемых дисциплин в рамках заочного и развивающегося дистанционного обучения.

11. Игры. Компьютерные игры широко используются не только для развлечения, но и как эффективное средство преодоления психологического барьера при первоначальном освоении и развитии навыков работы с компьютером. С целью обучения широко используются развивающие игры, ориентированные на определенную предметную область.

3.Программирование

3.1.Понятие программы и программирования

Программирование – деятельность по составлению программ.
Программа – это описание алгоритма решения задачи, заданное на языке ЭВМ.
Команда – предписание, определяющее очередной шаг (рис.3.1.).
Операция – это то, что должна сделать ЭВМ согласно каждой команде.
Операнды – это участники операции, то над чем и с чем выполняется операция.

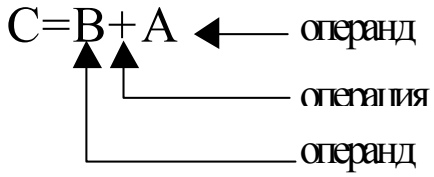


Рис.3.1. Схема представления команды

Операции, в которых участвуют два операнда, называются бинарными, а операции, в которых участвует один операнд - унарными (например, логическое отрицание).

Набор элементарных операций и способов их описаний образуют систему команд языка программирования.

3.2.Виды программного обеспечения

Программное обеспечение (ПО) – совокупность программ, позволяющих организовать решение задач на ЭВМ.

ПО и архитектура ЭВМ (аппаратное обеспечение) образуют комплекс функциональных средств ЭВМ, определяющих способность решения какого-либо класса задач. Различают программное и математическое обеспечение. Математическое обеспечение – это математические методы и алгоритмы, обеспечивающие решение поставленных задач, но не являющиеся функциональной частью средств компьютера.

По назначению ПО делится в основном на четыре класса (рис.3.2.).

Пакеты прикладных программ представляют собой комплекс программ, предназначенных для решения определенного класса задач. Библиотеки - это часто используемые программы вычисления функций, решения уравнений, распространенные операции обработки данных, например сортировки и т.д. Прикладное ПО характеризуется развитым интерфейсом и функциональной завершенностью. Современные прикладные пакеты используют средства графического интерфейса, которые характеризуются наглядностью отображения наличных компьютерных ресурсов и большей естественностью взаимодействия с ними.

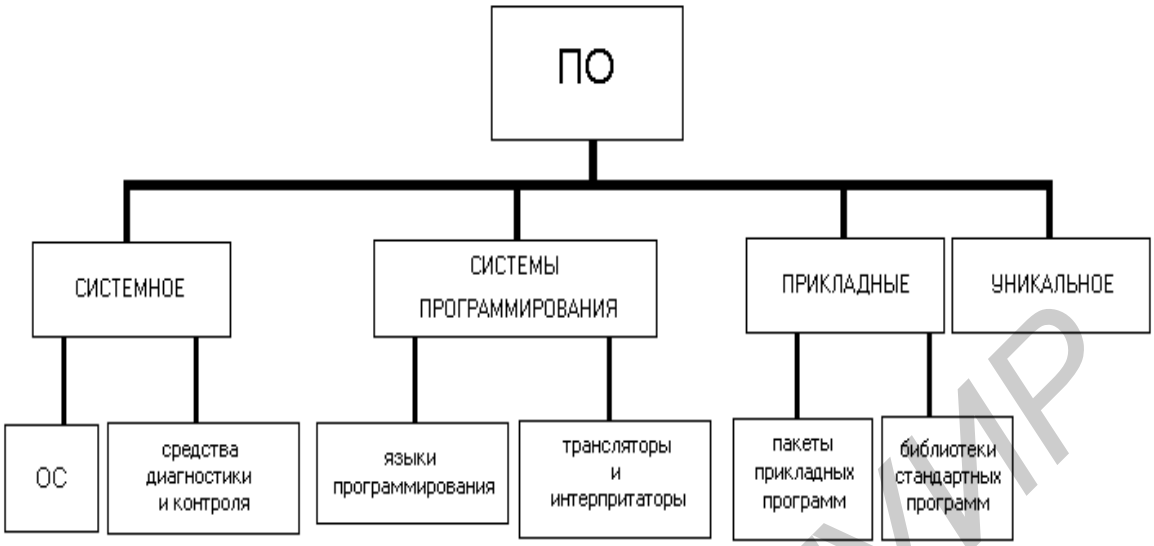


Рис.3.2.Структурная схема видов ПО

3.3.Разработка программ

3.3.1.Проектирование программ

Под проектированием программы будем понимать процесс преобразования постановки задачи, план алгоритмического или вычислительного решения с учетом реальных возможностей вычислительных машин и методов программирования.

Проектирование сверху вниз (нисходящее проектирование) начинается с наиболее абстрактного описания функций системы. По этому общему описанию верхнего уровня создаются более детальные описания. Процесс детализации продолжается до получения проекта, пригодного для программирования. Такой подход дает обозримое описание задачи на каждой стадии, а также представление взаимосвязи всех составных частей проекта и позволяет своевременно замечать возникающие проблемы и не переходить к последующей детализации до тех пор, пока полностью не завершится предыдущий уровень.

Результирующий проект имеет структуру дерева. Каждый уровень представляет собой законченное описание системы с конкретной степенью детализации.

Каждый пока еще не запрограммированный модуль заменяется при сборке программы заглушкой, которая удовлетворяет правилам интерфейса с головной программой, но не функций модуля, или выполняет их частично. На каждой стадии процесса реализации уже созданная программа должна правильно функционировать по отношению к более низкому уровню.

Поскольку каждая новая часть программы тестируется по мере включения в целое, то полностью реализованная программа является уже оттестированной. Такой способ объединения позволяет иметь работающую программу, выявлять и устранять ошибки на ранних стадиях программирования.

3.3.2.Структурное программирование

Структурное программирование – это метод разработки программы, поддерживаемый соответствующими технологиями, позволяющий лимитировать количество возможных ошибок в результирующей программе.

Как правило, под структурным программированием понимают программирование сверху вниз, начиная с модулей, требования к которым вытекают из требований к программе в целом.

Программирование сверху вниз начинается с самого высокого уровня, затем разрабатывается следующий уровень и объединяется в единую программу, которая тестируется и т.д. до тех пор, пока не будет включен и оттестирован нижний уровень реализуемой структуры.

Программирование снизу вверх (восходящее программирование) – это способ реализации и тестирования программ в обратной иерархической последовательности, начинающийся с модулей нижайшего уровня. Отладка обычно проводится с помощью специальных отладочных программ – от отдельных блоков самого низкого уровня до полного набора блоков.

На практике часто используются компромиссные решения, при которых сначала создается общая логическая структура программы, а затем разрабатываются часто используемые блоки низкого уровня, после чего применяется метод сверху вниз.

3.3.3. Модульное программирование

Модульное программирование представляет собой способ написания программ небольшими функционально законченными частями. Эти части называются модулями и обычно оформляются как подпрограммы или функции.

Каждый модуль содержит одну или несколько функций, входящих в задачу. Модульность позволяет упростить проектирование, чтение программ, проверку и их модификацию, поскольку легко устанавливает соответствие структурных единиц с выполняемыми функциями, а также создает библиотеки хорошо отлаженных программ.

В большинстве традиционных языков программирования модули выполняются как отдельно компилируемые программные блоки. Информационное взаимодействие между ними осуществляется через списки передаваемых параметров с помощью глобальных данных или общих областей.

3.3.4. Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) – основано на объектно-ориентированном подходе, базирующемся на введении абстракции «объект». Данные и поведение в ООП жестко связаны, они представлены в виде классов, экземпляры которых объекты. Например, некоторая формула может иметь диапазон допустимых значений, и на этих значениях могут быть определены, например, такие операции, как сложение и умножение. Вычисления в ООП рассматриваются как модель поведения.

Все объекты в ООП отвечают за свое поведение. Поведение проявляется как реакция на внешние раздражители (события). Каждый объект обладает определенным перечнем событий, на которые определена его реакция (черта поведения). Сведения о событии передаются сообщением, получаемым через интерфейс класса.

Внутренняя сложность объекта скрыта от пользователя, что реализуется как ограничение на доступ к деталям реализации класса. Такая конструкция класса обеспечивается механизмом инкапсуляции, определенным в ООП. Классы в ООП образуют иерархическую структуру, на которой определен механизм наследования.

Наследование предполагает возможность построения новых (производных) классов из существующих, называемых базовыми. При этом повторно используется уже имеющийся программный код.

Наиболее мощным механизмом, определенным в ООП, является полиморфизм, предполагающий множественность форм реализации функции или оператора. Соответствующий код функции в программе на C++ вызывается на основе ее сигнатуры, определяемой списком типов аргументов в перечне параметров функции. При иерархическом наследовании используются также виртуальные функции-члены класса, которые позволяют выбирать соответствующий код реализации во время выполнения программы.

3.4.Понятие алгоритма и его свойства

Определение алгоритма и его свойства. Алгоритм – это точное, т.е. сформулированное на определенном языке конечное описание последовательности действий, необходимых для выполнения некоторой работы или для решения конкретной задачи.

Свойства алгоритма:

1. Дискретность – разбиение алгоритма на ряд отдельных законченных действий (шагов).
2. Точность – это указание последовательности шагов.
3. Понятность – каждый исполнитель алгоритма должен однозначно написать и быть в
4. состоянии выполнить каждый шаг алгоритма.

5. Результативность – получение результата за конечное число шагов.
6. Массовость – применимость алгоритма к решению целого класса задач.
7. Детерминированность (определенность) – однозначность результата процесса решения при заданных исходных данных.
8. Способы представления алгоритма:
9. Формульная запись y = (2-x)+(3x-5).
10. Табличная запись.
11. Словесная запись.

Рассмотрим на примере описание алгоритма сортировки с помощью прямого включения. При этом алгоритме элементы массива мысленно делятся на уже отсортированную последовательность $a_1,\ldots,a_{i-1}$ и исходную не отсортированную последовательность. При каждом шаге, начиная с i=2 и увеличивая i каждый раз на единицу, из исходной последовательности извлекается i-й элемент и перекладывается в нужное место готовой, т.е. отсортированной последовательности. В процессе поиска подходящего места удобно, чередуя сравнения и движения по последовательности, как бы просеивать i-й элемент, т.е. сравнивать его с очередным элементом a_j, а затем либо этот элемент вставляется на свободное место, либо a_j сдвигается (передается) вправо и процесс “уходит” влево. Для работы этого алгоритма понадобится дополнительный элемент массива, поэтому необходимо расширить диапазон индекса в описании переменной a.

Запись на алгоритмическом языке:

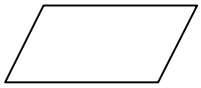
```
ПРОЦЕДУРА Прямое Включение.
ПЕРЕМЕННЫЕ a[0..n]: массив элементов. i, j: индексы. x: элемент.
НАЧАЛО
  ДЛЯ  i=2 ДО n ДЕЛАТЬ
    x = a[i]. a[0] = x. j = i.
    ПОКА x < a[j-1] ДЕЛАТЬ
      a[j] = a[j-1]. j = j – 1.
    КОНЕЦ.
    a[j] = x.
  КОНЕЦ
КОНЕЦ Прямое Включение.
```

Графический способ представления алгоритма (блок-схема). Блок-схема – графическое представление алгоритма, в котором каждое элементарное действие представляется в виде специальной графической фигуры (блока). Последовательность выполнения действий изображается линиями и стрелками, соединяющими эти блоки.

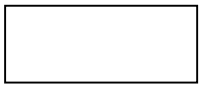
Под блоком понимается любой конечный этап из всего вычислительного процесса.

Типы блоков:

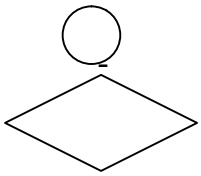
- начальный и конечный блок;



- информационный блок;



- функциональный блок;



- соединительный блок;

- блок проверки условия.

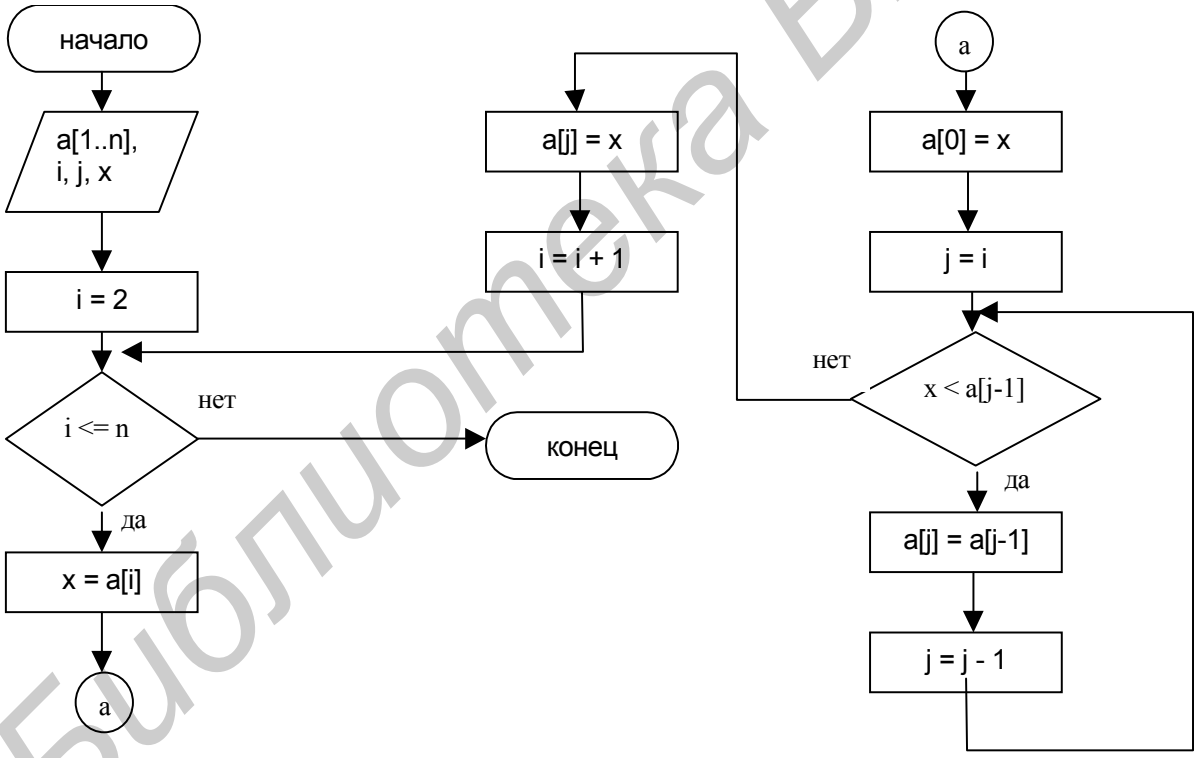


Рис.3.3. Пример алгоритма сортировки массива с помощью прямого включения

4.Основы программирования на Си

4.1.Язык Си и разработка программ

В 1972 году сотрудник фирмы Bell Labs Денис Ритчи создал язык Си во время совместной деятельности с Кеном Томпсоном над операционной системой UNIX. Прообразом Си послужил язык Би, разработанный Томпсоном.

Со временем язык Си становится одним из наиболее мощных и популярных языков. Причиной этого являются следующие достоинства:

Си - современный язык. Его структура побуждает программиста использовать в своей работе методы: нисходящее проектирование, структурное программирование и модульную структура программ.

Си - эффективный язык. Программы на Си отличаются компактностью и быстротой исполнения. Си – переносимый или мобильный язык, т.е. программа перенесена с небольшими изменениями (или вообще без них) на другую.

На Си программа, написанная для одной вычислительной системы, может быть перенесена с небольшими изменениями (или вообще без них) на другую.

Си – мощный и гибкий язык. Например, большая часть операционной системы UNIX написана на Си (компиляторы и трансляторы других языков: Фортран, АПЛ, Паскаль, Лого, Бейсик).

Программы, написанные на Си, используются для решения задач различных уравнений. Си обладает рядом мощных конструкций ассемблера.

Си - удобный язык. Он слишком структурирован и вместе с тем, не слишком ограничивает свободу программиста.

Си – язык компилирующего типа.

Чтобы заставить компьютер выполнить что-нибудь полезное, необходимо составить соответствующую программу.

Программа - это последовательность инструкций, реализующих алгоритм, набор предписаний, однозначно определяюший содержание и последовательность выполнения операций для решения задачи.

Например: чтобы вычислить значение простейшего выражения a=b+cd, где b, c и d могут принимать целочисленные значения, допустим в интервале от 0 до 100, необходимо выполнить следующие инструкции:

1. Ввести значение b, c и d.
2. Умножить c на d и сохранить значение произведения, например в f.
3. Сложить b со значением полученного произведения и результат присвоить a.
4. Вывести результат вычисления значения.

На языке Си эта программа будет выглядеть следующим образом:

```
# include  <stdio.h>                //подключение файла описания стандартных
                                    //программ ввода/вывода.
                                    // символ «//» - признак строчного комментария.
```

```
void main (void )          //заголовок головной функции программы.
{ int a,b,c,d,f;           //объявление переменных целого типа.
scanf ("%d %d %d",&b,&c,&d);  //вызов функции ввода данных scanf, и передача
                           // ей параметров: строки управления вводом. (в кавычках) и списка
                           //адресов переменных, задаваемых через запятую. Функция принимает и
                           //присваивает значение переменным при их поступлении с клавиатуры.
f=c*d;                     //вычисление произведения и сохраняет результат в f.
a= b+f ;                   //вычисляется сумма и заносится в a.
printf("%d", a);           //вызов стандартной функции распечатки
                           //данных. Распечатывается значение переменной a, со
                           //спецификацией  %d ,  т.е. как целое число.
}
```

Если данная программа подготовлена для выполнения, то пользователю остается:

- запустить программу на выполнение;
- при остановке (ожидание ввода) ввести через пробел значения b_c _d, например, набрать числа через пробел: (10 25 18) и нажать клавишу ВВОД (Enter);
- получить на экране дисплея результат (просмотреть можно, нажав ALT+F5 или используя меню windopw->user screen);

Но для подготовки программы необходимо (рис.4.1.):

- составить ее в одном из языков программирования;
- транслировать ее в стандарте одного из языков программирования;
- связать с необходимыми вспомогательными программами и функциями;
- загрузить в оперативную память;
- и, наконец, выполнить и получить результаты.

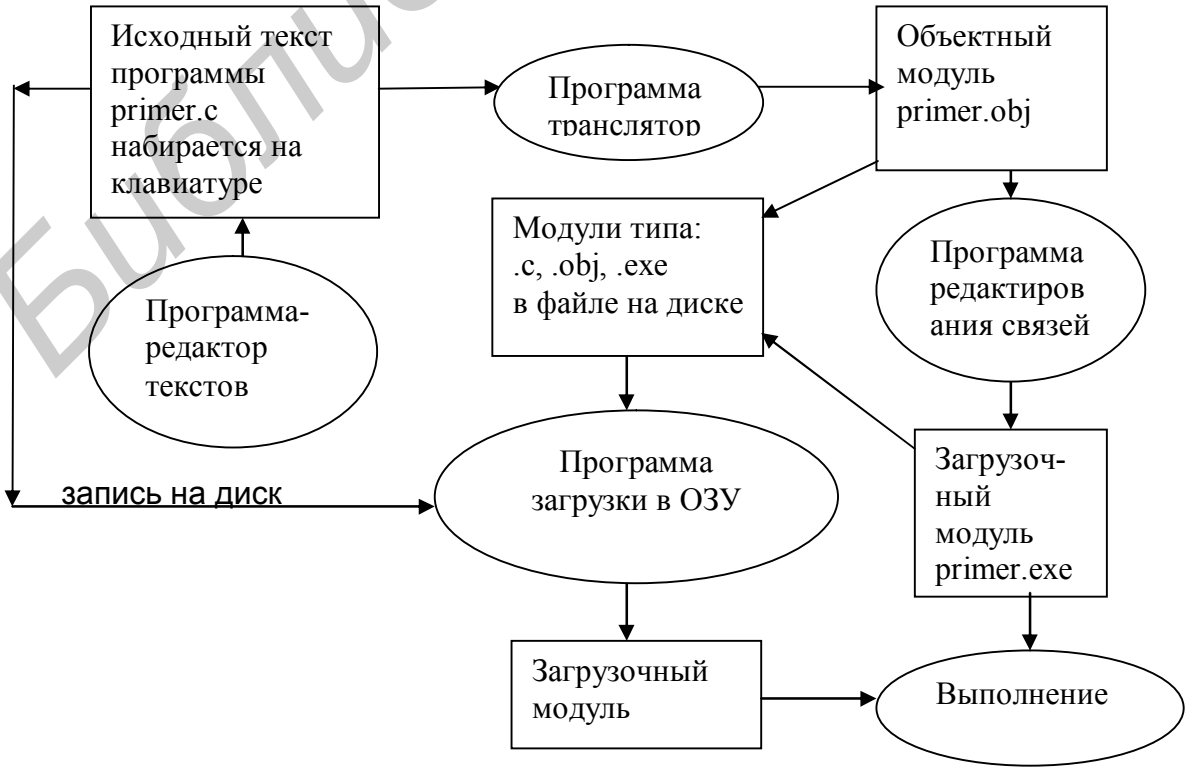


Рис.4.1.Схема процесса компиляции

Как правило, разработка таких программ осуществляется на основе инструментальных языков и систем программирования, к которым относятся:

- ассемблеры и макроассемблеры;
- трансляторы с языков высокого уровня;

4.2.Элементы программирования

4.2.1.Понятие идентификатора

Идентификатор используется в качестве имени объекта (функции, переменной, константы и т.п.). Идентификаторы должны выбираться с учетом следующих правил:

1) обязательно начинаться с буквы латинского алфавита (a,..., z. A,..., Z) или с символа подчеркивания (_), в них могут использоваться буквы латинского алфавита, символ подчеркивания и цифры (0,...,9) . Использование других символов в идентификаторах запрещено;

2) буквы нижнего регистра (a, ..., z), применяемые в идентификаторах, отличаются от букв верхнего регистра (A, ..., Z). Это означает, что следующие идентификаторы считаются разными: prog, ProG, PROG, pRoG и т.п.;

3)идентификаторы могут включать любое число символов, из которых воспринимаются и используются для выявления различных объектов (имен) только первые 32.

4.2.2.Типы данных и объявление переменных

Программа оперирует с различными данными, которые могут быть простыми и структурированными. Простые данные - это целые и вещественные числа, текст и указатели (содержат адреса памяти, по которым размещаются данные). В языке различают понятия описание переменной и ее определение (объявление). Описание устанавливает свойства объекта: его тип, размер и т.д. Определение наряду с этим вызывает выделение памяти. Каждый тип данных определяется одним из следующих ключевых слов:

Таблица 4.1

Название типа	Тип значения переменной	Диапазон значений	Необходимая память, в битах	Примечания
1	2	3	4	5

Название типа	Тип значения переменной	Диапазон значений	Необходимая память, в битах	Примечания
1	2	3	4	5
Int	Целый	-32768 ...32767	16	Задает значения, к которым относятся все целые числа, например –6, 0, 28 и т.д.

Окончание таблицы 4.1 .

1	2	3	4	5
short	Короткий и Целый	-32768 ...32767	16	Объекты short не могут быть больше, чем int. В Borland C int и short равной длины
long	Длинный и Целый	214748364… 2147483647	32	Используется, когда диапазон значений выходит за пределы диапазона типа int
char	Символьный	Символы кодовой таблицы ASCII (0...255)	8	Задает значения, которые представляют различные символы, например w, у, ф, 4, !, ., * и т. д. Этот тип часто используется как наименьшее беззнаковое целое значение
unsigned	Беззнаковый			Модификатор типов char, short, int, long, определяющий их беззнаковыми[1)]
float	Вещественный	±3.4e-38… ±3.4e+38	32	Определяет вещественные числа, дробная часть которых отделяется точкой (например, -5.27, 0.0, 31.69 и т.д.). Вещественные числа могут записываться в экспоненциальной форме. Например: -1.58e+2 (что равно -1,58 * 10^2), 3.61e-4 (что равно 3,61 *10^{-4}).
double	двойная точность	±1.7e-308... ±1.7e+308	64	Определяет вещественные переменные двойной точности, занимающие в два раза больше памяти, чем переменная типа float

[1)]Типы unsigned (например unsigned char) могут принимать большие по абсолютной величине положительные значения, чем переменные знаковых типов, за счет использования знакового бита для представления числа. Например, переменная типа unsigned int может принимать значения от 0 до 65535 (просто int при том же размере в битах от –32768 до 32767). По умолчанию unsigned a определяется как unsigned int a.

Примеры объявления данных:

int a, b;

unsigned i, j;

float k;

Здесь объявлены переменные: целые a и b , беззнаковые целые i и j , вещественное число одинарной точности k.

4.3.Локальные и глобальные переменные

В языке С переменные делятся на глобальные и локальные.

Глобальные переменные объявляются в файле исходного текста программного модуля вне какой-либо из функций (локальные объявляются внутри функции). Глобальные переменные создаются в точке объявления и доступны (видимы) в исходном тексте от точки объявления до конца файла, в котором они объявлены (они видимы и внутри функций). Глобальные переменные видимы также и для внешних модулей (см. прил. 3).

Локальные переменные по отношению к функциям являются внутренними. Они начинают существовать в точке объявления внутри функции и уничтожаются при выходе из нее. Если они записаны в списке параметров функции (в круглых скобках), то следует рассматривать такое объявление как введенное до первой открывающейся фигурной скобки. Для тех локальных переменных, которых нет в списке параметров, объявление делается после первой открывающейся фигурной скобки.

В среде Borland C++ объявление можно записать в любом месте программного кода функции. Объявленная в функции переменная является видимой от точки объявления до конца блока операторов (закрывающей фигурной скобки), в котором она объявлена. Здесь под блоком операторов понимается множество операторов, ограниченное фигурными скобками.

4.4.Ввод – вывод информации

В С имеется ряд функций, предназначенных для реализаций операций ввода-вывода. Наиболее используемая – функция форматированного вывода:

printf(“управляющая строка вывода“, список_переменных_через_запятую);

Формат printf включает в себя как текстовые сообщения, так и управляющие символы. Управляющим символам предшествует символ %, за которым могут следовать буквы, определяющие прототип вывода значений переменных. Выбор прототипа зависит от типа переменной, значение которой будет выводиться вместо прототипа. Основные прототипы переменных перечислены в табл.4.2.

Таблица 4.2

Название типа	Формат	Примечание
char	%c	
char[n]	%s	(Строка - массив символов), где n – количество символов в строке.
Int	%d	
long	%ld	
float	%f	
double	%lf	

Количество форматов в маске ввода должно соответствовать количеству переменных в списке переменных после кавычек. Переменные разделяются между собой запятыми. В формат могут входить также специальные символы, приведенные в табл.4.3.

Таблица 4.3.

Символ	Назначение
\n	Новая строка
\t	Табуляция
\\	Вывод символа \
\”	Вывод символа “

Символы, не являющиеся символами формата или спецсимволами, непосредственно выводятся функцией printf.

Пример использования оператора printf для вывода значений переменных a,b:

```
#include <stdio.h>                    // подключение библиотеки stdio.h
                                      // с функциями ввода-вывода
void main(void)                       // основная функция main
{
        int a,b;
        a=5;   b=10;                  // объявление переменных a,b
        printf("a = %d ,a b = %d;\n", a, b);   // вывод значений переменных a,b
}                                     //в форме a=5,a b=10;
```

Оператор ввода предназначен для ввода значений переменных с клавиатуры. Формат оператора scanf соответствует формату оператора printf. Отличие заключается в

том, что перед значениями переменных всех типов, за исключением массивов (строк символов), ставится амперсанд – символ “&.”. Он означает, что в распоряжение функции предоставляется не содержимое, а адрес переменной, что будет рассмотрено в разделе изучения указателей.

scanf(“формат“,X1,…Xn);

Пример использования оператора scanf для ввода значений переменных a,b:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{
  int a,b;                          // объявление переменных a,b
  scanf ( "%d%d", &a, &b );         // ввод значений переменных a,b с
//клавиатуры осуществляется путем набора этих значений через пробел и
//нажатия клавиши «ВВОД» (“Enter”).
  printf("a = %d  b = %d\n", a, b);     // вывод значений переменных a,b
}
```

4.5.Языковые средства ветвления

Все выражения, реализующие условия в конструкции выбора, должны заключаться в круглые скобки.

Логические операции. В языке С для работы с логическими операторами приняты несколько основных вариантов обозначения операций сравнения, которые представлены в табл.4.4.

Таблица 4.4

Обозначение	Операция
!=	Не равно
==	Равно
<	Меньше
>	Больше
<=	Меньше равно
>=	Больше равно
&&	Логическое И (исполняется, если все условия выполнены)
||	Логическое ИЛИ (исполняется, если хотя бы одно условие выполнено)

Если необходимо проверять несколько условий, каждое условие берется в свои скобки, а между скобками ставятся логические операторы (в зависимости от логики). В случае выполнения нескольких условных операторов, эти операторы берутся в фигурные скобки. Обратите внимание, что перед оператором не ставится точка с запятой.

Секция выполняется каждый раз при проходе цикла.

4.5.1.Оператор if

Синтаксис оператора if:

if (выражение) оператор;

Там, где синтаксис языка предписывает использовать оператор, может стоять и составной (блок операторов, заключенный в фигурные скобки), и пустой оператор (символ «;» - точка с запятой). Если выражение в заголовке условного оператора вырабатывает ненулевое значение, то оператор в условном операторе выполняется, в противном случае управление передается оператору, следующему за условным. Пример:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{       int a;                      // объявление переменных a
        scanf ( "%d", &a);          // ввод значений переменных a с клавиатуры
        if(a==3)                    // сравнение  переменной a с 3
                printf( "a равно 3");// вывод сообщения на экран в случае
                                    // выполнения условия
}
```

4.5.2.Конструкция if else

Синтаксис оператора if else таков:

```
if ( выражение )
оператор1;
else
оператор2;
```

Если значение выражения не равно нулю, то выполняется оператор1, в противном случае - оператор2. Пример:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{       int a;                      // объявление переменных a
        scanf ( "%d", &a);          // ввод значений переменных a с клавиатуры
        if(a==3)                    // сравнение  переменной a с 3
        printf( "a равно 3");       // вывод сообщения на экран в случае
```

```
                                        // выполнения условия
    else printf( "a не равно 3");       // вывод сообщения на экран в случае
                                        // не выполнение условия
}
```

4.5.3.Условная операция ?

Условная операция **?** может с успехом использоваться вместо конструкции if else там, где входящие в нее операторы являются простыми выражениями. Синтаксис условной операции таков:

результат = выражение ? выражение1 : выражение2;

Для примера рассмотрим программу. Переменной result при ее инициализации будет присвоено значение b, если выражение (a < 0) истинно, и a, если выражение (a < 0) ложно. В примере значение переменной result зависит от введенного значения переменной a.

```
#include <stdio.h>                          // подключение библиотеки
                                            // stdio.h
void main(void)                             // основная функция main
{       int a,b=0;                          // объявление переменных a
        scanf ( "%d", &a);                  // ввод значений переменных a с
                                            // клавиатуры
        int result = ( a < 0 ) ? b : a;     // объявление переменной result
                                            // по условию
        printf( "a = %d  b = %d result = %d\n", a, b );// вывод значений
                                            // переменных a,b, result
}
```

4.5.4.Оператор switch

Конструкция switch заменяет разветвленный многократный оператор if else. Синтаксис оператора switch таков:

```
switch ( выражение ) {
case константное_выражение_1 :
оператор(ы) case константное_выражение_2 :
оператор(ы)
case константаое_выражение_3 :
оператор(ы) default:
оператор(ы)
```

}

После вычисления выражения в заголовке оператора его результат последовательно сравнивается с константными выражениями, начиная с самого верхнего, пока не будет установлено их соответствие. Тогда выполняются операторы внутри соответствующего case, управление переходит на следующее константное выражение, и проверки продолжаются. Именно поэтому в конце каждой последовательности операторов должен присутствовать оператор break. После выполнения последовательности операторов внутри одной ветки case, завершающейся оператором break, происходит выход из оператора switch. Обычно оператор switch используется тогда, когда программист хочет, чтобы была выполнена только одна последовательность операторов из нескольких возможных.

Каждая последовательность операторов может содержать нуль или более отдельных операторов. Фигурные скобки в этом случае не требуются.

Ветка, называемая default (умолчание), может отсутствовать. Если она есть, то последовательность операторов, стоящая непосредственно за словом default и двоеточием, выполняется только тогда, когда сравнение «выражение» ни с одним из стоящих выше константных выражений (в case)не истинно. Пример:

```
#include <stdio.h>                          // подключение библиотеки stdio.h
void main(void)                             // основная функция main
{       int a;                              // объявление переменных a
        scanf ( "%d", &a);                  // ввод значений переменных
                                                // a и с клавиатуры
        switch(a)
        {       case 3: printf( "a равно 3");   // вывод сообщения на экран в случае a=3
                case 4: printf( "a равно 4");   // вывод сообщения на экран в случае a=4
                default:
                printf( "a = %d\n", a);         // вывод значения переменной a
        }
}
```

4.5.5.Оператор goto

Оператор goto используется для безусловной передачи управления внутри функции от одного оператора к другому. Синтаксис оператора goto:

goto идентификатор;

Управление передается на оператор в теле функции, помеченной указанным идентификатором. Пример:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{       int a;                      // объявление переменных a
       scanf ( "%d", &a);           // ввод значений переменной  a  с клавиатуры
```

```
    if(a>=0) goto label1;          // переход на метку label1, если a>=0
    a=0;                           // присвоение переменной a значения 0
    label1:printf( "a = %d\n", a); // вывод значения переменной a
}
```

Хотя в языке С и разрешена передача управления на любой оператор в теле функции, опыт показывает, что следует пользоваться этой возможностью как можно реже или вовсе от нее отказаться. В соответствии с теорией структурного и объектно-ориентированного программирования использование оператора goto нежелательно, так как может затруднить возможности и свести на нет усилия компилятора по оптимизации программы.

Если все-таки применяется оператор goto, то целесообразно придерживаться следующих рекомендаций:

- не входите внутрь блока извне;
- не входите внутрь оператора if или else конструкции if else или оператора switch;
- не входите внутрь итерационной структуры (оператора цикла) извне этой структуры.

4.6.Циклы

Циклы, или итерационные структуры, позволяют повторять выполнение отдельных операторов или групп операторов. Число повторений в некоторых случаях фиксировано, а в других определяется в процессе счета на основе одной или нескольких проверок условий.

Циклы завершаются в следующих случаях:

- обратилось в нуль условное выражение в заголовке цикла.;
- в теле цикла выполнился оператор break;
- в теле цикла выполнился оператор return.

В первых двух случаях управление передается на оператор, располагающийся непосредственно за циклом. В третьем случае происходит возврат из функции (завершение работы данной функции).

Бывают циклы с проверкой условия перед началом выполнения тела цикла (top-testing), по окончании выполнения тела (bottom-testing) или внутри тела (middle-testing). Ниже рассмотрены все указанные типы циклов.

4.6.1.Цикл while

Синтаксис цикла while (пока):

while (условное_выражение) оператор

Ясно, что в цикле типа while проверка условия производится перед выполнением тела цикла (оператор). Если результат вычисления условного выражения не равен нулю, то выполняется оператор (или группа операторов). Перед входом в цикл while в первый раз

обычно инициализируют одну или несколько переменных для того, чтобы условное выражение имело какое-либо значение. Оператор или группа операторов, составляющих тело цикла, должны, как правило, изменять значения одной или нескольких переменных, входящих в условное выражение, чтобы в конце концов выражение обратилось в нуль, и цикл завершился.

Потенциальной ошибкой при программировании цикла while, как, впрочем, и цикла любого другого типа, является запись такого условного выражения, которое никогда не прекратит выполнение цикла. Такой цикл называется бесконечным (например цикл: while (a) printf("Circle"), где a - любое число, отличное от 0. Цикл будет бесконечно выводить на экран дисплея текст Circle) . Пример:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{       int a;                      // объявление переменных a
        scanf ( "%d", &a);          // ввод значений переменных a и с клавиатуры
        while(a>=0)                 // цикл повторяется пока a>=0
        { printf( "a = %d\n", a);   // вывод значения переменной a
            a--;                    // уменьшение значения переменной a на один
        }
}
```

4.6.2.Цикл do while

В цикле do while проверка условия осуществляется после выполнения тела цикла. Синтаксис цикла:

```
do  оператор;
        //тело цикла
while ( условное_выражение )
```

В языке Си вместо одиночного оператора (например в теле рассматриваемого цикла) может быть подставлена группа операторов (блок). Цикл while прекращает выполняться, когда условное выражение обращается в нуль (становится ложным). Пример:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{       int a;                      // объявление переменных a
        scanf ( "%d", &a);          // ввод значений переменных a и с клавиатуры
        do{                         // начало цикла
            printf( "a = %d\n", a); // вывод значения переменной a
            a--;                    // уменьшение значения переменной a на 1
        } while(a>=0);              // цикл повторяется пока a>=0
}
```

4.6.3.Цикл for

Наиболее общей формой цикла в языке С является цикл for. Цикл for - это более общая и более мощная форма, чем аналогичный цикл в языках Паскаль и Бейсик.
Конструкция for выглядит следующим образом:

for (выражение1; выражение2; выражение 3) оператор;

Каждое из трех выражений можно опускать. Хотя в принципе каждое из этих выражений может быть использовано программистом как угодно, обычно первое выражение служит для инициализации индекса, второе -для выполнения проверки на окончание цикла, а третье выражение - для изменения значения индекса.
Формально это правило можно описать так:
1. Если первое выражение присутствует, то оно вычисляется.
2. Вычисляется второе выражение (если оно присутствует). Если вырабатывается значение 0, то цикл прекращается, в противном случае цикл будет продолжен.
3. Исполняется тело цикла.
4. Вычисляется третье выражение (если оно присутствует).
5. Выполняется переход к п.2.

Выполнение в любом месте тела цикла оператора continue приводит к немедленному переходу к шагу 4. Пример:

```
#include <stdio.h>                  //подключение библиотеки stdio.h
void main(void)                     //основная функция main
{       int a;                      //объявление переменных a
        for(a=0; a<10; a++)         //цикл от 0 до 9 -й переменной по
        printf( "a = %d\n", a);     // вывод значения переменной a
}
```

Цикл for можно свести к циклу while следующим образом:
Цикл for:

for (выражение1; выражение2; выражение3) оператор;

переводится в:

```
выражение1;
while ( выражение2 ) {
оператор;
выражение3;          }
```

4.7.Функции

Процесс разработки программного обеспечения предполагает расчленение сложной задачи на набор более простых подзадач. В Языке С поддерживаются функции как

логические единицы (поименованные блоки текста программы), служащие для реализации отдельных подзадач.

Функции в С должны иметь уникальные имена. Существенное значение имеет тип функции, точнее тип возвращаемого значения (функция может возвращать только одно значение) и принимает ли функция параметры (список формальных параметров функции объявляется в круглых скобках после имени функции и содержать 0, 1, или несколько параметров).

Простейшим примером использования функций является функция main (головная), которая должна присутствовать в любой программе, разрабатываемой на С. С нее начинается выполнение программы. Чаще всего main функция не имеет принимаемых параметров и не возвращает значения. В этом случае вместо отсутствующих типов указывается ключевое слово void (если тип возвращаемого параметра не указан, то по умолчанию тип возвращаемого значения такой функции является int). Пример: void main(void).

Если функция возвращает значение, то это значение передается вызвавшему фрагменту при помощи записываемого в операторе «return» выражения. Return вызывает завершение работы данной функции, передачу значения в функцию, вызвавшую данную, и возврат управления на оператор, следующий после вызова функции.

При вызове функции указываются фактические параметры вызова, их количество должно соответствовать числу и типу формальных параметров в заголовке вызываемой функции. Если функция не возвращает значения (т.е. возвращает void), то оператор return может использоваться в варианте «return;» (без следующего за ним выражения). Пример:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
float pi(void)
{                                  // функция только возвращает значение константы ПИ
    return  3.14159265359;         // возвращаемое значение
}
int SummaAandB(int A,int B)
{                                  //  функция возвращает сумму переменных A и B
    return A+B;                    // возвращаемое значение
}
void PrintName(void){      //функция выводит на экран имя Serg 10 раз по одному
                           //в строке, не возвращает и не принимает никаких значений
  for(int t=0;t<10;t++)
     printf("Serg\n");
}
void PrintA(int A){        //  функция выводит на экран значение переменной A
   printf("%d",A);
```

```
}
void main(void){
   float f=pi();                    // вызов функции pi
   printf("pi=%f\n",f);
   int i=SummaAandB(1,3);           // вызов функции SummaAandB
   printf("summa(A+B)=%d\n",i);
   PrintName();                     // вызов функции PrintName
   PrintA(20);                      // вызов функции PrintA
}
```

4.8.Указатели

В языке Си существует два способа доступа к переменной: ссылка на переменную и использование механизма указателей. Указатель - переменная (указатель) - переменная, предназначенная для хранения адреса в памяти. Указатель - константа - значение адреса ОП.

Определены две операции для доступа к переменным через указатели: " & " и " * ";
операция & - присвоить значение адреса;
операция * - выбрать содержимое из адреса.

Признаком переменной-указателя для компилятора является наличие в описании:
1) типа объекта, для доступа к которому используется указатель;
2) символа * перед переменной:

```
int var1, *prt;
```

Такое объявление приводит к появлению переменной var1 типа int и указателя на тип int, т.е. "указатель на целое". Место, выделяемое для него транслятором, зависит от модели памяти (может быть 2 или 4 байта).

Унарная операция & дает возможность присвоить адрес переменной указателю, т.е. y= &x, присваивает адрес x как содержимому переменной указателя y (рис.4.2).

Операцию & можно применять только к переменным и элементам массива. Недопустимы y=&(x+7), &25 .

Унарная операция * воспринимает свой операнд как адрес некоторого объекта и использует его для выборки содержимого, если y = &x; z = *y; => z = x; (рис4.3)
Указатели могут встречаться в выражениях, как и любая другая переменная. Допустимы выражения:
*y = 7; - в ячейку с адресом *y занести 7;
x = 5; - содержимое с адресом *x увеличить в пять раз;
(*z)++; - добавить 1 к содержимому с адресом *z.

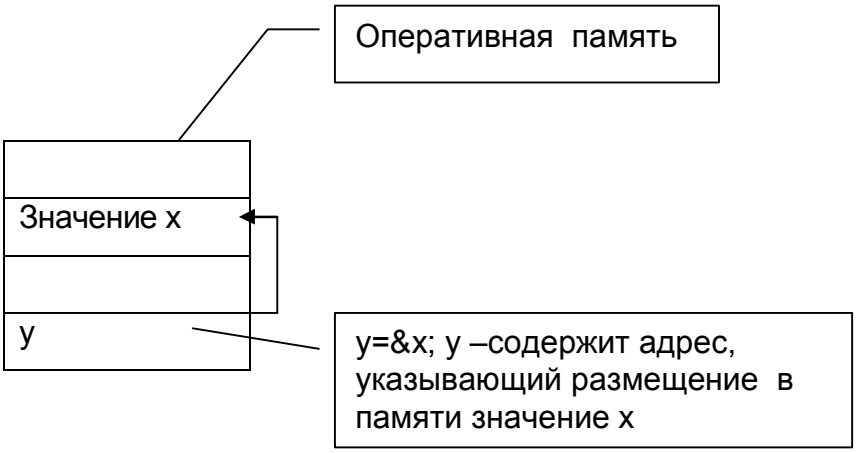


Рис.4.2.Схема образования ссылки на переменную

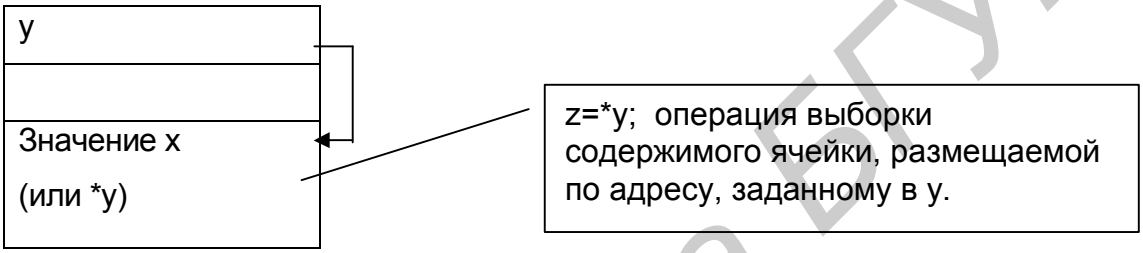


Рис.4.3.Схема извлечения значения по ссылке на переменную

Характерно, что указатели можно использовать в арифметических операциях. Например: если - указатель, то операция y++ увеличивает его значение на одну длину типа указателя. Транслятор будет масштабировать приращение адреса в соответствии с типом, заданным в объявлении указателя. После выполнения операции y++, y будет указывать на следующий элемент данного типа. Это свойство в языке Си используется при работе со структурированными данными.

4.9.Программа в Си, связь между функциями и передача параметров в функцию

Программы на языке Си обычно состоят из некоторого числа отдельных функций (подпрограмм), среди которых должна быть одна с именем main. С этой функции начинается выполнение программы. Как правило, функции имеют небольшие размеры, и могут находиться как в одном, так и в нескольких файлах. Если функции располагаются в различных физических файлах, то для выполнения их как единой программы, необходимо собрать их в файле проекта. В языке Си запрещено определять одну функцию внутри другой, поэтому все имена функций являются глобальными. Связь между функциями осуществляется через аргументы, возвращаемые значения и внешние (глобальные) переменные. Передача значения (возврат значения) из вызванной функции в вызвавшую реализуется с помощью оператора возврата, который записывается в следующем формальном виде:

return выражение;

Таких операторов в подпрограмме может быть несколько, и тогда они фиксируют соответствующие точки выхода. Вызвавшая функция может при необходимости игнорировать возвращаемое значение. После слова return можно ничего не записывать. В этом случае вызвавшей функции никакого значения не передается. Управление передается вызвавшей функции и в случае выхода "по концу" без использования return (последняя закрывающаяся фигурная скобка). Примеры:

```
#include <stdio.h>
int f1 (void)  {printf("rabotaet f1()"); return 1;} //функция возвращает значение 1
void main (void)  {int k=f1();} // возвращаемое значение используется
                                //во внешней программе

#include <stdio.h>
int  f1 (int a,int b)  {return a+b;} //функция принимает параметры и возвращает значение
void main (void)
{int a=17; int b=16;printf("%d",f1(a,b));} //возвращаемое значение распечатывается в main

#include <stdio.h>
f1 (int a,int b)  {a+b;}                //функция принимает параметры
                                        // и возвращает значение по умолчанию
void main (void)                        //результат работы тот же, что и в предыдущем случае
{int a=17; int b=16;printf("%d",f1(a,b));} //хотя программа некорректна, о чем выдается
//предупреждение, как и в варианте строки: {int a=17,b=16, c=f1(a,b); printf("%d",c);
```

Пример использования глобальных переменных:

```
#include <stdio.h>
#include <conio.h>
int a,b;                //глобальные переменные
int f1 (int x)          //x – локальный формальный параметр, принимает значение
{return (a+b)*x;}       //фактического параметра k при вызове функции f1(k);
void main (void)
{clrscr();int k=35; a=10;b=12; int c= f1(k);  printf("%d",c);  } //c и k– локальные в main
```

Если некоторые переменные, константы, массивы, структуры объявлены как глобальные, то их не надо включать в описок параметров вызванной функции. Она все равно получит к ним доступ.

Программе main также могут быть переданы параметры при запуске. В этом случае текст программы может выглядеть следующим образом:

```
#include <stdio.h>
```

```
void main(int argc, char *argv[])
 //здесь в argc – количество элементов в строке запуска программы;
{char *s;  s=argv[0];          // argv[] – массив ссылок на элементы этой строки
printf("name programm=\"%s\"\n",s);                    //первый элемент строки запуска
if(argc > 1)            //конструкция \" используется для ввода в текст символа кавычки «"»
{s=argv[1];  printf("parametr programm=\"%s\"\n",s);} //второй элемент строки запуска
}
```

Если создать загрузочный модуль такой программы, например с именем f1.exe, в каталоге D:\BORLANDC\MY_WORK и запустить его из этого каталога командой, например: f1 77, то в результате получим вывод на экран двух строк:

name programm=" D:\BORLANDC\MY_WORK \F1.EXE"

parametr programm="77"

В языке С аргументы функции передаются **по значению**, т.е. вызванная функция получает в именах формальных параметров временную копию каждого аргумента, а не его адрес. Это означает, что в самой функции не может изменяться значение самого оригинального аргумента в вызвавшей ее программе. Ниже будет показано, как убрать это ограничение. Если же в качестве аргумента функции используется имя массива, то передается начало массива (адрес начала массива), а сами элементы не копируются. Функция может изменять элементы массива, сдвигаясь (индексированием) от его начала. Рассмотрим, как соответствуют друг другу параметры в вызове и в списке функции. Пусть в вызывающей программе вызов функции реализован следующим образом:

```
int a,b=73, c=40;     a = fun(b,c);
```

Здесь b и o - аргументы, значения которых передаются в вызываемую подпрограмму. Если описание вызываемой функции начинается так:

void fun(int b,int c);

то имена передаваемых аргументов в вызове и в программе fun будут одинаковыми. Если же описание функции начинается, например, строкой void fun(int i,int j); , то вместо имени b в вызвавшей функции для того же аргумента в функции fun будет использовано имя i, а вместо c – j .

Пусть обращение к функции имеет вид

```
a = fun(&b,&c);
```

Здесь подпрограмме передаются адреса переменных b и o. Поэтому прототип (заголовок вызываемой функции) должен быть, например, таким:

```
void fun(int *k,int *c);
```

Теперь k получает адрес передаваемой переменной b, а c - адрес передаваемой переменной c. В результате в вызвавшей программе c - это переменная целого типа, а в вызванной программе c - это указатель на переменную целого типа. Если в вызове записаны те же имена, что и в списке параметров, но они записаны в другом порядке, то все равно устанавливается соответствие между i-м именем в списке и i-м именем в вызове.

Выше уже указывалось, что переменные передаются функции по значению, поэтому нет прямого способа в вызванной функции изменить некоторую переменную в вызвавшей функции. Однако это легко сделать, **если передавать в функцию не переменные, а их адреса.** Пример:

```
#include <stdio.h>
void f1 (int * a,int b, int * c1)//принимаемые аргументы: a – по адресу,  b – по значению,
{*c1=*a+b; return;}          //c1- по адресу. Теперь указатель c1 ссылается на c
void main (void)             // и значения c=*c1;
{int c, a=17,  b=16; f1(&a,b,&c);//передаваемые параметры: a – по адресу; b – по значентию
printf("%d",c);       }      // c – по адресу. Результат вычисления получаем в c.
```

Рассмотрим теперь, как функции можно **передать массив** в виде параметра. Здесь возможны три варианта. Причем в прототипе и теле записи могут комбинироваться из этих вариантов.

int mn[100] – объявлен массив. В функцию передается (прототип)
int mn []
int *mn

1. Параметр задается как массив (например: int [100]).
2. Параметр задается как массив без указания его размерности (например: int m[ ]).
3. Параметр задается как указатель (например: int *m, прототип int f1(int *mn).

Независимо от выбранного варианта вызванной функции указатель передается на начало массива. Сами же элементы массива не копируются.

```
#include <stdio.h>
void f1 (int x[20],int k)
{int i;  for (i=0; i<k; i++) x[i]=i;}
#define k 5
void main (void)
{ int a[k],i; f1(a,k);
for (i=0; i<k; i++)  printf("%d;",a[i]);  }
```

Вызываемая функция может быть записана и в виде

```
void f1 (int x[],int k)  {int i;  for (i=0; i<k; i++) x[i]=i;} //или
void f1 (int * x,int k)  {int i; for (i=0; i<k; i++) x[i]=i;}
```

Функции в языке С необходимо объявлять. Мы будем использовать уже рассмотренные ранее конструкции, называемые прототипом. В соответствующем объявлении будет дана информация о параметрах. Она представляется в следующем виде: тип функция (параметр_1,параметр_2,...,параметр_n);

Для каждого параметра можно указать только его тип (например, тип функции (int, float);), а можно дать и его имя (например, тип функция (int a,float b);). В языке С разрешается создавать функции с переменным числом параметров. Тогда при задании прототипа вместо последнего из них указывается многоточие (например void f1(int a, float b,…).

Можно определить и функцию void f1 (...) {тело функции}и обращение f1 (a1, a2), где a1 и a2 достаются каким-либо особым образом из стека или известны как глобальные).

4.10.Структурированные типы данных

4.10.1.Перечисление

В языке С предусмотрена возможность использования особого типа, позволяющего создавать переменные, принимающие только определенные значения (константы, определяемые при создании типа). Этот тип является не чем иным, как перечислением возможных значений, принимаемых переменными данного типа. Декларация типа осуществляется словом enum, после которого в фигурных скобках перечисляются значения, принимаемые данным типом.

Пусть идентификатор времена года (seasons) может принимать одно из четырех значений: весна, лето, осень, зима (spring, summer, autumn, winter), тогда пример будет выглядеть так:

enum seasons{spring, summer, autumn, winter} p,w;

В примере переменные p,w могут принимать одно из четырех значений времени года. После введения типа seasons можно объявлять переменные (объекты) данного типа, например enum seasons a,b,c;

Введем еще одно объявление:

enum days {mon, tues, wed, thur, fri, sat,sun} my_week;

Имена, занесенные в days, представляют собой константы целого типа. Первая из них (mon) автоматически устанавливается в нуль, и каждая следующая имеет значение на единицу больше, чем предыдущая (tues=1, wed=2 и т.п.). Можно присвоить константам определенные значения целого типа (именам, не имеющим их, будут, как и раньше, назначены значения предыдущих констант, увеличенные на единицу). Например:

enum days {mon=5, tues=8, wed=10, thur, fri, sat} my_week;

После этого mon=5, tues=8, wed=10, thur=11, fri=12, sat=13, sun=14.

Тип enum можно использовать для задания констант true = 1 false = 0, например enum t_f {false true} a, b;

К сожалению, контроль за соответствием значений констант, присваемых переменным типа enum, не осуществляется.

```
void main (void)
{enum my_en {a, b=10, c,d};
my_en  i1, i2,i3;   //введенный тип my_en  можно использовать для объявления переменных
enum vit {r, l, m=10} j1, j2;
// правильные действия
i1=a;   i2=b;   i3=a;   j1=j2;
```

```
//некорректные действия, действия выполняются но вызывают выдачу сообщений
i1=0;  i1=555;  i1=r;
//ошибочные действия
//  a=b;  a=j1;  a=i1;    a=0;    b=10;
}
```

4.10.2.Массивы

В программе на языке С можно использовать структурированные типы данных. К ним будем относить массивы, структуры и файлы.

Массив состоит из многих элементов одного и того же типа. Ко всему массиву целиком можно обращаться по имени. Кроме того, можно выбирать любой элемент массива. Для этого необходимо задать индекс, который указывает относительную позицию элемента в массиве. Число элементов массива назначается при его объявлении и в дальнейшем не меняется. Если массив объявлен, то к любому его элементу можно обратиться следующим образом : указать имя массива и индекс элемента в квадратных скобках. Массивы объявляются так же, как и переменные :
int a [100]; массив a из 100 элементов целого типа : a[0], a[1],…., a[99] (индексация всегда начинается с нуля).
char b [30];
float c [42]; - элементы массива b имеют тип char, а c – float.

Двухмерный массив представляется как одномерный, элементы которого тоже массивы. Например, объявление char a[10][20]; задает двумерный массив символов. По аналогии можно установить и большее число измерений. Элементы двухмерного массива хранятся по строкам, т.е. если проходить по ним в порядке их расположения в памяти, то быстрее всего изменяется самый правый индекс. Например, обращение к девятому элементу пятой строки запишется так: a[5][9]. Пусть задано объявление: int a[2][3]; Тогда элементы массива a будут размещаться в памяти следующим образом: a[0][0], a[0][1] a[0][2], a[1][0], a[1][1], a[1][2]. Имя массива a – это указатель константы, которая содержит адрес его первого элемента (для нашего примера – a[0][0]). Предположим, что a=1000. Тогда адрес элемента a[0][1] будет равен 1002 (элемент типа int занимает в памяти 2 байта), адрес следующего элемента a[0] [2] – 1004 и т.п. Что же произойдет, если вы выберете элемент, для которого не выделена память. К сожалению, компилятор не следит за этим. В результате возникнет ошибка, и программа будет работать не верно.

В языке С существует сильная взаимосвязь между указателями и массивами. Любое действие, которое достигается индексированием массива, может быть выполнено и с помощью указателей, причем последний вариант будет быстрее. Объявление int a [5]; определяет массив из пяти элементов: a[0], a[1], a[2], a[3], a[4]. Если объект y объявлен как int *y; то оператор y=&a[0]; присваивает

переменной y адрес элемента a[0]. Если переменная y указывает на текущий элемент массива a, то y+1 указывает на следующий элемент, причем здесь выполняется соответствующее масштабирование для приращения адреса с учетом длины объекта (для типа int – 2 байта, long – 4 байта, double – 8 байт и т.п.). Поскольку само имя массива есть адрес его нулевого элемента, то инструкцию y=&a[0]; можно записать и в другом виде: y=a;. Тогда элемент a[i] можно представить как *(a+i). С другой стороны, если y – указатель, то следующие две записи y[i] и *(y+i) эквивалентны. Между именем массива и соответствующим указателем есть одно важное различие. Указатель – это переменная и y=a; или y++; -допустимые операции. Имя же массива – константа, поэтому конструкции вида a=y; a++; использовать нельзя, так как значение константы постоянно и не может быть изменено.

Переменные с адресами могут образовывать некоторую иерархическую структуру (могут быть многоуровневыми), типа указатель на указатель (то есть он содержит адрес другого указателя), указатель на указатель на указатель и т.п. Их использование будет рассмотрено ниже на примере.

Если указатели адресуют элементы одного массива, то их можно сравнивать (отношения вида: <, >, = =, ! = и др. работают правильно).

В то же время нельзя сравнивать либо использовать в арифметических операциях указатели на разные массивы (соответствующие выражения не приводят к ошибкам при компиляции, но в большинстве случаев не имеют смысла). Как и выше, любой адрес можно проверять на равенство или неравенство со значением NULL. Указатели на элементы одного массива можно также вычитать. Тогда результатом будет число элементов массива, расположенных между уменьшаемым и вычитаемым объектами.

Язык С позволяет инициализировать массив при объявлении. Для этого используется такая форма:

тип имя_массива [ ] [ ] = { список значений };

Рассмотрим примеры:

```
int a[5] = {0,1,2,3,4};
char c[7] = { 'a','b','c','d','e','f','g'};
int b [2] [3]= {1,2,3,4,5,6};
```

В последнем случае: b[0][0] = 1, b[0][1] = 2, b[0][2] = 3, b[1][0] = 4, b[1][1] = 5, b[1][2] = 6.

В языке допускаются массивы указателей, которые объявляются, например, следующим образом: char *m[5]; . Здесь [m] – массив, содержащий адреса элементов типа char.

4.10.3.Строки символов

Язык С не поддерживает отдельный строковый тип данных, но он позволяет определить строки двумя различными способами. В первом используется массив символов, а во втором – указатель на первый символ массива. Объявление char a[10]; указывает компилятору на необходимость резервирования места для максимум 10 символов. Константа а содержит адрес ячейки памяти, в который помещено значение первого из десяти объектов типа char. Процедуры, связанные с занесением конкретной строки в массив а, копируют ее по одному символу в область памяти, на которую указывает константа а, до тех пор, пока не будет скопирован нулевой символ, оканчивающий строку. Когда выполняется функция типа printf("%s",a);, ей передается значение а, т.е. адрес первого символа, на который указывает а. Если первый символ нулевой, то работа функции printf заканчивается, а если нет, то она выводит его на экран, прибавляет к адресу единицу и снова начинает проверку на нулевой символ. Такая обработка позволяет снять ограничения на длину строки (конечно, в пределах объявленной размерности): строка может быть любой длины до тех пор, пока есть место в памяти, куда ее можно поместить.

Вторым способом определения строки является использование указателя на символ. Объявление char *b; задает переменную b, которая может содержать адрес некоторого объекта. Однако в данном случае компилятор не резервирует место для хранения символов и не инициализирует переменную b конкретным значением. Когда компилятор встречает инструкцию вида b=“Москва”, он производит следующие действия. Во-первых, как и в предыдущем случае, создает в каком-либо месте объектного модуля строку Москва, за которой следует нулевой символ. Во-вторых, присваивает значение начального адреса этой строки (адрес символа М) переменной b. Функция printf(“%s”,b); работает так же, как и в предыдущем случае, осуществляя вывод символов до тех пор, пока не встретится заключительный нуль.

Массив указателей можно инициализировать, т.е. назначать его элементам конкретные адреса некоторых заданных строк при объявлении. Если объявить char my_char [3,7], в памяти выделится следущий блок памяти (рис.4.4).

Г	Р	И	Ш	А	!	\0
Э	Т	О	\0	\0	\0	\0
Я	\0	\0	\0	\0	\0	\0

Рис.4.4. Вид выделенного блока памяти при статическом выделении памяти

Если же объявить char * my-char [3] и инициализировать этими словами, то в памяти будет выделено, что значительно экономит память (рис.4.5).

Г	Р	И	Ш	А	!	\0
Э	Т	О	\0			
Я	\0					

Рис.4.5.Вид выделенного блока памяти при использовании указателя

Пример:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main(void)
{int l;  clrscr();
char str_array[3] [10]={"Гриша!","это","я"};  //инициализация 2-мерного массива
for (l=0; l<sizeof(str_array); l++)
printf("%c:",*(str_array[0]+l)); //распечатка 2-мерного массива символов
// для распечатки шестнадцатеричных кодов  2-мерного массива символов
// заменить оператор на
//printf("%x:",*(str_array[0]+l));
printf("\nDlina_2-mernogo_massiva=%d\n",sizeof(str_array));
char* point_array[3]={"Гриша!","это","я"}; //инициализация массива указателей
printf("Obschaja_dlina_massiva_ukazatelej_i_strok=%d\n",sizeof(point_array)
+strlen(point_array[0])+ strlen(point_array[1])+strlen(point_array[2]));
}
```

4.10.4.Структуры

Структура – это объединение одного или более объектов (переменных, массивов, указателей, других структур и т.п.). Как и массив, она представляет собой совокупность данных. Отличием является то, что к ее элементам необходимо обращаться по имени и что различные элементы структуры не обязательно должны принадлежать одному типу. Объявление структуры осуществляется с помощью ключевого слова struct, за которым идет ее тип и далее список элементов, заключенных в фигурные скобки:

```
struct тип {
тип_элемента_1 имя_элемента_1;
        …
 тип_элемента_n имя_элемента_n; };
```

Именем элемента может быть любой идентификатор. Как и выше, в одной строке можно записывать через запятую несколько идентификаторов одного типа. Рассмотрим пример:

```
struct date {int day;
        int month;
        int year;    };
```

Следом за фигурной скобкой, заканчивающей список элементов, могут записываться переменные данного типа, например: struct date {int day;int month;

int year;} a,b,c; В этом случае для каждой переменной выделяется память, объем которой определяется суммой длин всех элементов структуры. Описание структуры без последующего списка не вызывает выделения никакой памяти, а просто задает шаблон нового типа данных, которые имеют форму структуры. Введенное имя типа позже можно использовать для объявления структуры, например struct date days;. Теперь переменная days имеет тип date. При необходимости структуры можно инициализировать, помещая за объявлением список начальных значений элементов. Разрешается вкладывать структуры одна в другую, например:

```
struct man { char name [30], fam [20];
        struct date bd;
        int voz; };
```

Определенный выше тип data включает три элемента: day, month, year, содержащий целые значения (int). Структура man включает элементы: name[30], fam[20], bd и voz. Первые два name[30] и fam[20] – это символьные массивы из 30 и 20 элементов каждый. Переменная bd представлена составным элементом (вложенной структурой) типа data. Элемент voz содержит значения целого типа (int). Теперь разрешается объявить переменные, значения которых принадлежат введенному типу:

```
struct man _ man_ [100];
```

Здесь определен массив _man_, состоящий из 100 структур типа man. В языке С разрешено использовать массивы структур. Структуры могут состоять из массивов и других структур.

Чтобы обратиться к отдельному элементу структуры, необходимо указать ее имя, поставить точку и сразу за ней написать имя нужного элемента, например:

```
_man_ [i].voz = 16;
_man_ [j].bd.day = 22;
_man_ [j].bd.year =1976;
```

При работе со структурами необходимо помнить, что тип элемента определяется соответствующей строкой объявления в фигурных скобках. Например, _man_ имеет тип man, year - является целым числом и т.п. Поскольку каждый элемент структуры относится к определенному типу, его имя может появляться везде, где разрешено использовать значения этого типа. Допускаются конструкции вида _man_[i] = _ man_[j]; где _man_[i] и _man_[j] - объекты, соответствующие единому описанию структуры. Другими словами, разрешается присваивать одну структуру другой по их именам.

Унарная операция & позволяет взять адрес структуры. Предположим, что задано объявление

```
struct date { int d,m,y; } day;
```

Здесь day - это структура типа date, включающая три элемента: d,m,y. Другое объявление struct date _*db; устанавливает тот факт, что db - это указатель на структуру типа date. Запишем выражение: db = &day;. Теперь для выбора элементов d,

m, у структуры необходимо использовать конструкции: (*db).d, (*db).m, (*db).y. Действительно, db - это адрес структуры, *db - сама структура. Круглые скобки здесь необходимы, так как точка имеет более высокий, чем звездочка приоритет. Для аналогичных целей в языке С предусмотрена специальная операция ->. Она тоже выбирает элемент структуры и позволяет представить рассмотренные выше конструкции в более простом виде: db->d, db->m, db->y.

4.10.5.Битовые поля

Особую разновидность структур представляют поля. Поле - это последовательность соседних бит внутри одного целого значения. Оно может иметь тип signed int либо unsigned int и занимать от 1 до 16 бит. Поля размещаются в машинном слове в направлении от младших к старшим разрядам.
Например, структура

```
struct prim {int a:2; unsigned b:3; int :5;
             int c:1; unsigned d:5; }  f,j;
```

обеспечивает размещение, показанное на рис. 4.6. 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.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
d	d	d	d	d	b	не используется					b	b	b	a	a

Рис. 4.6.Пример блока памяти, выделенного под битовое поле

В полях типа signed крайний левый бит является знаковым. Например, такое поле шириной 1 бит может только хранить значения -1 и 0, так как любая ненулевая величина будет интерпретироваться как -1.

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 &. Пример:

```
#include <stdio.h>
struct  {unsigned a : 1;    unsigned b : 1;      unsigned y : 1;      unsigned c : 2; } f;
void main () {     int i;
   printf("размер f=%d байт \n",sizeof(f));
   f.a =f.b=1;    // в поля a и b записывается 1
   for (i=0;i<2;i++)
   {    f.y =f.a && f.b; //  коньюнкция a и b
        printf(" цикл %d;f.y =%d\n", i, f.y);
        f.b=0;      }
   f.c = f.a +!f.b;              // сложение значений f.a и отрицание f.b (=1)8)
```

```
printf("f.c=%d", f.c);}       // f.c=2
```
результаты работы:
размер f=1 байта
цикл o;f.y =1
цикл 1;f.y =0
f.c=2

4.10.6.Смеси

Смесь - это разновидность структуры, которая может хранить (в разное время) объекты различного типа и размера . В результате появляется возможность работы в одной и той же области памяти с данными различного вида. Для описания смеси используется ключевое слово union, а соответствующий синтаксис аналогичен синтаксису структуры.

Пусть задано объявление:

union r { int ir; float fr; char cr; } z;

Здесь ir имеет размер 2 байта, fr - 4 байта и cr - 1 байт. Для z будет выделена память достаточная, чтобы сохранять самый большой из трех приведенных типов. Таким образом, размер z будет 4 байта. В один и тот же момент времени в z может иметь значение только одна из указанных переменных (ir, fr, cr). Пример:

```
#include <stdio.h>
union r{ int ir; float fr; char cr;} z;
float f;
// объявл. смесь z типа r: .Размер смеси будет определяться размером самого
//длинного элемента, в данном случае fr,
void main (void)
{       //в версии Borland C++ версии 3.1 обнаружена ошибка при
        //использовании в вычислениях и преобразованиях вывода
        //вещественных значений элементов структур . Чтобы обойти ошибку,
        //выбираем вещественное значение элемента union в простую
        //вещественную переменную f (f=z.fr;), а затем используем f в
        //выражениях и наоборот.
printf ("размер z=%d байта \n",sizeof(z));
        // sizeof(z) вычисляет длину переменной z и printf распечатывает
        //вычисленную длину
printf ("введите значение z.ir \n");     //выдача приглашения для ввода
scanf  ("%d",&z.ir);                     //ввод целого значения в элемент z.ir
printf ("значение ir =%d \n",z.ir);      //вывод значения z.ir
printf ("введите значение z.fr \n");     //приглашение для ввода
                                          //вещественного значения
```

```
scanf  ("%f",&f);              //ввод вещественного значения в переменную f и
z.fr=f;          //запись в z.fr (фактически реализован ввод: scanf  ("%f",&z.ir);.
printf ("значение fr=%f \n",f);    //вывод значения вещественной переменной
printf ("введите значение z.cr \n");            // приглашение на ввод информации
flushall();                        // очистка буферов ввода-вывода.
        //Такая очистка буфера здесь необходима, т.к. в буфере ввода остается
        //символ конца строки от предыдущего ввода, который затем введется
        //спецификацией %c , вместо реально набираемого символа
scanf  ("%c",&z.cr);                    //чтение символа, введенного с клавиатуры
printf ("значение cr=%c;\n",z.cr);      //вывод значения символа
}
```

Пример сеанса работы с программой («Enter» - это нажатие этой клавиши):

размер z= 4 байта
введите значение z.ir
7 «Enter»
значение ir=7
Введите значение z.fr
38.345678«Enter»
Значение fr=8.345678
Введите значение z/cr
P«Enter»
Значение cr= P;

4.10.7. Директива typedef

Рассмотрим описание структуры

srtuct data { int d,m,y; };

Фактически мы ввели новый тип данных - data. Теперь его можно использовать для объявления конкретных экземпляров структуры,
например:

srtuct data a,b,c;

В язык С введено специальное средство, позволяющее назначать имена новым типам данных. Таким средством является оператор typedef. Он записывается в следующей общей форме:

typedef тип имя;

Здесь "тип" - любой разрешенный тип данных и "имя" - любой разрешенный идентификатор.
Рассмотрим пример:

typedef int INTEGER;

После этого можно сделать объявление:

INTEGER a,b;

Оно будет выполнять то же самое, что и привычное объявление:

int a,b;

Другими словами, INTEGER можно использовать как синоним ключевого слова int. При этом можно комбинировать объявления со словами int и INTEGER, например:

INTEGER a,b;

int o,d;

Здесь объявлены четыре переменные целого типа (фактически int a,b,c,d;).

4.11.Работа с указателями

Если объявить

int mas[100], *p, a;

то:

1) для массива отводится память в адресном пространстве под 100 элементов типа int.
2) память отводится под указатель-константу с именем MAS, значением указателя является адрес массива.
3) память отводится под указатель-переменную с именем p.

Операция инициализации указателя может осуществляться только операцией "присвоить адрес некоторой переменной".

p = &a;

p = &mas[0]; или p = mas;

или присвоением p = NULL;, где NULL - константа, определенная через define в файле NULL.H.

Допустимо p=0, но не рекомендуется.

Ошибкой являются:

a=10;

p=a; // где p - указатель. Присвоение невозможно, т.к. типы int* и int.

p=10; // присвоение невозможно, т.к. типы int* и const int.

Указателю нельзя присваивать целые значения, но можно складывать и вычитать указатель и целые числа.

p+=10; - экв. p = p+10; - увеличение адреса на 10* масштабный множитель.

p-=2; уменьшение на 2* масшт. множителя.

p+=10; увеличивает на 10 содержимое ячейки, на которую ссылается p.

Например:

Если p=mas; то p+=10; эквивалентно p=p+10 и эквивалентно присвоению p=&mas[10];

*p+=10; эквивалентно mas[0]=mas[0]+10;

Если 2 указателя ссылаются на элементы одного и того же массива, то допускаются операции отношения над ними: = =; !=; <, >, и т.д.

Для указателей, ссылающихся на элементы разных массивов, результат сравнения не определен.

Допускается вычитание указателей.

Например, разработаем функцию вычисления длины строки:

```
int strlen(char *s)
{       char *p = s;            // объявлен указатель и инициирован адресом
                        // массива символов.
        While(*p! = '\0')       p++;
        return p-s;
}
```

Данная функция возвращает значение типа int, т.е. длина строки не может превышать значения, которое представимо типом int (2-х байтным целым). Поэтому в реальных программах лучше пользоваться стандартной функцией strlen, которая описана в файле string.h. Эта функция умеет выбирать тип возврата в зависимости от модели памяти, используемой в программе. Для этого определен (define) тип ptrdiff_t в файле STDDEF.H.

4.12.Работа с памятью

Разработаем пример простой программы, распределяющей память. Функция static char allocbuf[ALLOCSIZE]; выделяет память из буфера static char allocbuf[ALLOCSIZE];, а функция void afree(char *p); освобождает память, распределенную в этом буфере. Память организована в виде стека, где базовым адресом основания стека является allocbuf, а на вершину стека указывает alloccp.

```
#include <stdio.h>          //функции ввода-вывода
#include <string.h>         //функция strlen()
#include <conio.h>          //функции ввода-вывода на консоль(clrscr())
#define ALLOCSIZE 10000     //препроцессорная директива определения
                            //константы как равной 10000
static char allocbuf[ALLOCSIZE];    //буфер памяти (массив символов allocbuf длиной
                                    //в 10000 символов
static char *allocp;  //указатель константа на тип char
char * alloc(int n)   //заголовок функции
{                           //если размер оставшейся свободной памяти
        if (allocbuf + ALLOCSIZE - allocp >=n) // не меньше запрашиваемого
                            //участка, то память выделяется.
  {     //printf("%d\n",allocp);
```

```
        allocp +=n;                 //                      alloccp
        //printf("%d\n",allocp);    //   распределенная
        return allocp;              //   память              свободная память
   }    else                        //
        return 0;                   //
}                                   // allocbuf             allocbuff+ALLOCSIZE

void afree(char *p)                 //
{
        if (p>=allocbuf && p < allocbuf + ALLOCSIZE)      //память освобождается,
        allocp=p;                   //если адрес начала объекта принадлежит
}                                   //нашей области allocbuf.                    void
main (void)
{ char str[255],*p[1000];   //буфер ввода с клавиатуры str[255] и массив
int a,i,j;                  //для хранения списка указателей *p[1000];
a=0;            i=0;
p[0]=NULL; i++;
while (((scanf("%s",str))!=EOF) && (p[i]=alloc(strlen(str)))!=0) //ввод строк
{       //пока не введен конец файла или исчерпан буфер свободной памяти.
 printf("adress=%d", p[i]);
 i ++;}
 i--;
 printf("Базовый адрес =%d", allocbuf);
 for (j=0; j<=i; j++)
        printf("\np[%d]=%d;",j,p[j]);
 for (j=i;j>=0;j--)
        {afree(p[j]); printf("\указатель стека =%d",p[j]);
        }
}
```

4.13.Файлы

Файл – это организованный набор данных , расположенных на внешнем носителе.

В файлах размещаются данные, предназначенные для длительного хранения. Каждому файлу присваивается используемое при обращении к нему уникальное имя. В языке С отсутствуют инструкции для работы с файлами. Все необходимые действия выполняются через функции, включенные в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.п. Эти устройства сильно отличаются друг от друга. Однако файловая

система позволяет преобразовывать их в единое абстрактное логическое устройство, называемое потоком. Существует два типа потоков: текстовые и двоичные.
Прежде чем читать или записывать информацию в файл, он должен быть открыт. Это можно сделать с помощью библиотечной функции fopen. Она берет внешнее представление файла (например C:\MY_FILE.TXT) и связывает его с внутренним логическим именем, которое используется далее в программах. Логическое имя – это указатель на требуемый файл. Его необходимо объявлять, и делается это, например, так:

```
FILE *lst;
```

Здесь FILE - имя типа, описанное в стандартном определении stdlo.h, 1st - указатель на файл. Обращение к функции fopen в программе производится так:
lst=fopen(спецификация файла, вид использования файла);
Спецификация файла может быть, например : C:\MY_FILE.TXT - для файла MY_FILE.TXT на диске C:; A:\MY_DIR\EX2_3.CPP - для файла EX2_3.CPP в поддиректории A:\MY_DIR и т.п. Вид использования файла может быть:
r - открыть существующий файл для чтения;
w - создать новый файл для записи (если файл с указанным именем существует, то он будет переписан)
a - дополнить файл (открыть существующий файл для записи информации, начиная с конца файла, либо создать файл, если он не существует);
pb - открыть двоичный файл для чтения;
wb - создать двоичный файл для записи;
ab - дополнить двоичный файл;
rt - открыть текстовой файл для чтения;
wt - создать текстовой файл для записи;
at - дополнить текстовой файл;
r+ - открыть существующий файл для записи и чтения;
w+ - создать новый файл для записи и чтения;
a+ - дополнить или создать файл с возможностью записи и чтения;
r+b - открыть двоичный файл для записи и чтения;
w+b - создать двоичный файл для записи и чтения;
a+b - дополнить двоичный файл с предоставлением возможности записи и чтения.

Если режим t или b не задан (например, r, w или a), то он определяется значением глобальной переменной _fmode. Если _fmode = 0_BINARY, то файлы открываются в двоичном режиме, а если _fmode =O_TEXT - в текстовом режиме. Константы 0_BINARY и O_TEXT определены в файле fcntl.h.
Строки вида r+B можно записывать и в другой форме: rb+. Если в результате обращения к функции fopen возникает ошибка, то она возвращает указатель на константу NULL. После окончания работы с файлом, он должен быть закрыт. Это делается с помощью библиотечной функции fclose. Она имеет следующий прототип:

int fclose(FILE *lst);

При успешном завершении функция fclose возвращает значение нуль. Любое другое значение говорит об ошибке.

4.13.1.Вывод информации в файл

```
#include <stdio.h>

void main (void)
{
        char str[50];
        FILE  *rstr, *wstr, *pstr, *astr;
        rstr = fopen ("c:\\my_file.txt", "rt");
        wstr = fopen ("c:\\out_file.txt", "wt");
        pstr = fopen ("prn", "wt");
        astr = fopen ("c:\\out_plus.txt", "at");
        while (fscanf (rstr, " %S ", str) !=EOF)
        {
            printf ( " Вывод на дисплей: %S\n", str);
            fprintf (wstr, "%S\n", str);/*запись файла (прежнее содержание стирается)*/
            fprintf (pstr, "%S\n", str); /* вывод на печать*/
            fprintf (astr, "%S\n", str); /*дополнение файла*/
        }
        fclose(rstr); fclose(wstr); fclose(pstr); fclose (astr);
}
```

В данном примере указатели не инициализируются адресами соответствующих файлов, открытых для указанного типа операций.

Имя “prn”, используемое для вывода на печать, представляет собой стандартное имя устройства печати.

4.13.2.Чтение строк из файла и вывод их на экран

```
#include <stdio.h>
void main (void)
{
        char     str [50];
        FILE * fr, * fw;
        if ((fr=fopen ("A:\\fail.ttt","r+" ))==NULL)
        //открытие файла с дискеты
```

```
        {
                printf("Файл не открылся. \nВведите информацию с клавиатуры");
                fgets (str ,49, stdin); // можно gets (str ,49);
        }
        else
        fgets (         str ,49, fr);           // или введите строку       до 49
        printf ("Вывод строки: %s", str);       // символов без пробела
        if ((fw=fopen ("a:\\1.txt", "w+"))==NULL)
        {   printf("Файл не открылся");  }
        else
        {   printf("\n в файл 1.txt "); //запись в файл
            fputs (str, fw);            // функция записывает
                return;
         } //выход из программы
        // если файл не открылся, то вывод из str в файл ошибок.
        fprintf (stderr, " Вывод в стандартный файл для ошибок\n%s",str);
        fclose (fr); fclose (fw);
}
```

Программа считывает из файла fail.ttt дискеты, вставленной в дисковод A: 49 символов или пока не встретится символ конец строки. Если файл не открылся, то предлагает ввести информацию с клавиатуры (введется 48 символов или до нажатия клавиши Ввод). Затем информация выводится в файл 1.txt на дискете или, если не удалось его открыть, в файл ошибок на экран.

4.13.3.Библиотечные функции для работы с файлами

Рассмотрим теперь некоторые другие библиотечные функции, используемые для работы с файлами (все они описаны в файле stdio.h).

1. Функция putc записывает символ в файл и имеет следующий прототип:

int putc(int c,FILE *lst);

здесь lst - указатель на файл, возвращенный функцией fopen, с - символ для записи (переменная о имеет тип int, но используется только младший байт). При успешном завершении putc возвращает записанный символ, в противном случае возвращается константа EOF. Она определена в файле stdlo.h и имеет значение –1.

2. Функция getc читает символ из файла и имеет следующий прототип:

int getc (FILE *lst);

здесь 1st - указатель на файл, возвращенный функцией fopen. Эта функция возвращает прочитанный символ. Соответствующее значение определяется типом int, но старший байт равен нулю. Если достигнут конец файла, то getc возвращает значение EOF.

3. Функция feof определяет конец файла при чтении двоичных данных и имеет следующий прототип*:

int feof(FILE *lst);

здесь lst - указатель на файл, возвращенный функцией fopen. При достижении конца файла возвращается ненулевое значение, в противном случае возвращается 0.

4. Функция fputs записывает строку символов в файл. Она отличается от функции puts только тем, что в качестве второго параметра должен быть записан указатель на переменную файлового типа. Рассмотрим пример: fputs("Example",lst); При возникновении ошибки возвращается значение EOF.

5. Функция fgets читает строку символов из файла. Она отличается от функции gets только тем, что в качестве второго параметра указывается количество байт и в качестве третьего параметра должен быть записан указатель на переменную файлового типа. Рассмотрим пример: fgets(str,nlst); Функция возвращает указатель на строку при успешном завершении и константу NULL в случае ошибки либо достижении конца файла; (char*str, int n, file*lst читает n символов из строки или до конца строки).

6. Функция fprintf выполняет те же самые действия, что и функция printf, но работает с файлом. Ее отличием является то, что в качестве первого параметра задается указатель на переменную файлового типа. Рассмотрим пример:

fprintf(lst,"%x",a);

7. Функция fscanf выполняет те же самые действия, что и функция scanf, но работает с файлом. Ее отличием является то, что в качестве первого параметра задается указатель на переменную файлового типа. Рассмотрим пример:

fscanf(lst,"%x",&a);.

При достижении конца файла возвращается значение EOF.

8. Функция fseek позволяет выполнять чтение и запись с произвольным доступом и имеет следующий прототип:

int fseek(FILE *lst, long count, int access);

здесь lst - указатель на файл, возвращенный функцией fopen, count - номер байта относительно заданной начальной позиции, начиная с которого будет выполняться операция, access задает начальную позицию. Переменная access может принимать следующие значения.

0 - начальная позиция задана в начале файла; SEEK_SET (0)
1 - начальная позиция считается текущей; SEEK_CUR (1)
2 - начальная позиция задана в конце файла. SEEK_END

При успешном завершении возвращается нуль, при ошибке - ненулевое значение;

9. Функция ferror позволяет проверить правильность выполнения последней операции при работе с файлом и имеет следующий прототип:

int terror(FILE *lst);

В случае ошибки возвращается ненулевое значение, в противном случае возвращается нуль.

10. Функция remove удаляет файл и имеет следующий прототип:

int remove(char *file_name);

здесь file_name - указатель на строку со спецификацией файла. При успешном завершении возвращается нуль, в противном случае возвращается ненулевое значение.

11. Функция rewind устанавливает указатель текущей позиции в начало файла и имеет следующий прототип:

void rewind(FILE *lst).

В языке С открываются пять стандартных файлов со следующими логическими именами:

stdin - для ввода данных из стандартного входного потока (по умолчанию с клавиатуры);

stdout - для вывода данных в стандартный выходной поток (по умолчанию на экран дисплея)

stderr - файл для вывода сообщений об ошибках (всегда связан о экраном дисплея);

stdpm - для вывода данных на принтер;

stdaux - для ввода и вывода данных в коммуникационный канал.

Литература

1. Фигурнов В.Э. Программное обеспечение персональных ЭВМ. – М.: Наука, 1988г
2. Гукин Д. Word for Windows для начинающих: Пер. с англ. – Киев.: Диалектика, 1994г
3. Бемер С., Фратер Г. MS Access для пользователя: Пер. с нем. – Киев.: Торгово-издат. Бюро BHV, 1994г
4. Николь Наташа, Албрехт Ральф. Электронные таблицы Excel 5.0: Практ. пособие/ - М.: ЭКОМ., 1994г
5. П.Нортон. Программно- аппаратная организация IBM PC. Пер.с англ.-Москва, : Радио и связь, -1992г
6. Керниган Б. Ритчи Д. Язык программирования Си. – М.: Финансы и статистика, 1985г
7. Уэйт М., Прата С., Мартин Л, Язык Си. – М.: Мир, 1988г
8. Бруно Бабе. Просто и ясно о Borland C++: Пер. с англ. – М. Бином.,1992г
9. Касаткин А.И., ВальвачевА.Н. От TURBO C к Borland C++. Мн.: Выш. шк., 1992г

Приложение 1

Требования к содержанию пояснительной записки

Рекомендуемое содержание пояснительной записки КР и содержание каждого пункта:

1. «Введение» (1...2 листа) – содержит краткое описание вопросов, рассмотренных в курсовой работе;
2. Ответ на первый вопрос (3...5 листов) – содержит подробный ответ на первый вопрос задания контрольной работы;
3. Ответ на второй вопрос (3...5 листов) - содержит подробный ответ на второй вопрос задания контрольной работы;
4. Описание структуры программы, созданной на С (2...4 листа) – определяется структура программы, перечень и описание функций (см.приложение 4);
5. Описание функций программы (2...3 листа) – описываются логически законченные части программы, приводятся описание функций и блок-схемы разработанных алгоритмов, описываются принцип функционирования программы и структура пользовательского интерфейса (как и какие действия должен осуществлять пользователь для управления программой);
6. Выводы (1...2 листа) – описывает проделанную работу и функциональные возможности программы: возможность ввода \ вывода, характерные особенности программы, введенные вами новшества, ограничения на применение данной программы, требования к программному обеспечению и компьютеру, на котором будет осуществляться демонстрация;
7. Литература (не менее 5 источников, на которые должны быть ссылки в содержательной части пояснительной записки);
8. Приложение - содержит распечатку программы и, возможно, иллюстрации и схемы,вынесенные за пределы текста пояснительной записки.

Приложение 2

Требования к оформлению пояснительной записки

Общие положения. Оформление пояснительной записки выполняется в соответствии с требованиями стандарта принятыми в БГУИР, а также требований к оформлению программной документации Гост ЕСПД - единой системы программной документации.

Все листы пояснительной записки, кроме титульного, должны иметь сквозную нумерацию. Номер листа пишется в центре листа сверху.

Листы пояснительной записки должны быть сброшюрованы в скоросшивателе или в обложке из плотной чертежной бумаги (лист формата А3 согнут пополам). Одна половина обложки используется в качестве титульного листа, оформленного в соответствии с приложением 3.

Текст пояснительной записки должен распечатываться шрифтом **«Times New Roman»**, размер шрифта **12,** через **1.5** интервала.

Лист «Содержание» пояснительной записки размещают на отдельной (пронумерованной) странице (страницах), снабжают заголовком "СОДЕРЖАНИЕ", не нумеруют как раздел и включают в общее количество страниц пояснительной записки (см. содержание данного методического пособия).

В содержание пояснительной записки включают номера разделов, подразделов, пунктов и подпунктов, имеющих заголовки, их наименования и номера страниц; номера и наименования (при наличии) приложений пояснительной записки и номера страниц (см. оформление содержания данного методического пособия). Наименования, включенные в содержание, записывают строчными буквами. Прописными буквами должны записываться аббревиатуры. Пример возможных пунктов содержания пояснительной записки приведен в приложении 1.

Структурными элементами текста пояснительной записки являются разделы, подразделы, пункты, подпункты и перечисления. Внутри подразделов, пунктов и подпунктов могут быть даны перечисления, которые рекомендуется обозначать арабскими цифрами со скобкой: I), 2) и т.д. Допускается выделять перечисления записью дефиса перед текстом.

Заголовки разделов пишут прописными буквами и размещают симметрично относительно правой и левой границ текста. Заголовки подразделов записывают с абзаца строчными буквами (кроме первой прописной). Переносы слов в заголовках не допускаются. Точку в конце заголовка не ставят. Если заголовок состоит из двух предложений, их разделяют точкой. Каждый раздел рекомендуется начинать с нового листа.

Разделы, подразделы, пункты и подпункты следует нумеровать арабскими цифрами с точкой. Разделы должны иметь порядковый номер (1,2 и т.д.).

Иллюстрации. Иллюстрации могут быть расположены в тексте пояснительной записки и (или) в приложениях. Иллюстрации, если их в пояснительной записке более одной, нумеруют арабскими цифрами в пределах всего документа. В приложениях иллюстрации нумеруются в пределах каждого приложения в порядке, установленном для основного текста документа. Ссылки на иллюстрации дают следующим образом: "рис. 12" или "(рис.12)". Ссылки на ранее упомянутые иллюстрации дают с сокращенным словом "смотри", например, "см. рис. 12" (примеры оформления смотри в данном пособии).

Ссылки. В пояснительной записке допускаются ссылки на используемые документы и литературу (смотри ссылки в тексте данного пособия).

При ссылках на документ допускается проставлять в квадратных скобках его порядковый номер в соответствии с перечнем ссылочных документов. Допускается указывать только обозначение документа и (или) разделов без указания их наименований. Ссылки на отдельные подразделы, пункты и иллюстрации другого документа не допускаются. Допускаются ссылки внутри пояснительной записки на пункты, иллюстрации и отдельные подразделы.

Таблицы. Цифровой материал для достижения лучшей наглядности и сравнимости показателей, как правило, следует оформлять в виде таблицы.

Таблица может иметь заголовок, который следует выполнять строчными буквами. Прописными должны записываться заглавные буквы и аббревиатуры.

Сноски к таблицам располагают непосредственно под таблицей (смотри текст данного пособия).

Приложения. Иллюстративный материал, таблицы или вспомогательный текст допускается оформить в виде приложений. Приложения оформляют как продолжение пояснительной записки на последующих страницах.

Каждое приложение должно начинаться с новой страницы с указанием в правом верхнем углу слова «ПРИЛОЖЕНИЕ» прописными буквами и иметь тематический заголовок, который записывают симметрично тексту прописными буквами. При наличии в пояснительной записке более одного приложения все приложения нумеруют арабскими цифрами (без знака №). Например: ПРИЛОЖЕНИЕ 1, ПРИЛОЖЕНИЕ 2 и т.д. Все приложения должны быть перечислены в листе «Содержаниение» (смотри оформление данного пособия).

Приложение 3

Образец оформления титульного листа

Министерство образования Республики Беларусь
БГУИР
Заочный факультет
Кафедра экономической информатики

Контрольная работа по курсу
«Основы информатики и вычислительной техники»

вариант №___

Исполнитель:

Иванов И.И.
гр.901400
3 курса ЗФ,

Руководитель:

Петров П.П.

Минск

1998

Приложение 4

Пример разработки программы для контрольной работы

ЗАДАНИЕ: На фирме ведется файл учета персонала. Разработать интерфейсные средства (программу) поддержки ведения файла, позволяющие:
а) добавлять записи в массив структур;
б) осуществлять сохранение результатов выполнения программы в файле и считывания их из файла;
в) выводить результаты на экран в виде таблицы.

ОБЩИЕ РЕКОМЕНДАЦИИ:

Необходимо помнить, что в языке программирования С символы А и а – воспринимаются как разные символы. Символы русского алфавита нельзя использовать в названиях переменных и функций. Для включения шрифтов русского языка (или перехода с русского на английский) неоходимо нажать правую кнопку Ctrl на клавиатуре компьютера (настройка компьютерного центра БГУИР) либо попробывать вместе нажать комбинацию Shift+Alt, левый и правый Shift, Shift+Ctrl и т.д., либо спросить у лаборанта или владельца компьютера комбинацию клавиш переключения шрифтов. Допустимо выводить в программу русские слова латинскими буквами.

Основные этапы выполнения третьего пункта контрольной работы

Выполнение контрольной работы состоит из нескольких этапов:
1) создания базовой структуры;
2) создание файла данных;
3) определение структуры программы и выделение ее основных частей (функций);
4) непосредственное кодирование функций на языке С и создание программы;
5) компиляция и отладка программы.

1. Создание базовой структуры

Для создания базовой структуры необходимо определить, какие данные используются в программе. Выделим основные поля, которые должна содержать базовая структура. Пусть в данном случае в файле хранятся фамилия (текст), должность (текст) и возраст работника фирмы. Тогда записи файла данных могут быть представлены следующим образом:

Иванов мастер 34

Петров инженер 28

Определим типы полей базовой структуры. Базовая структура для представленного файла может быть представлена в виде:

```
struct Man{                    //  имя нового типа (структуры)
```

```
    char Name[12];                  //   поле, содержащее фамилию
    char Profession[20];            //   поле, содержащее фамилию
    int Age;                        //   поле, содержащее фамилию
};
```

2. Создание файла данных

Пусть файл данных должен быть сформирован на диске С в каталоге WORK в файле 1.dat (путь к файлу C:\WORK\1.dat). Создадим любыми доступными средствами каталог WORK на диске С.

Сформируем текстовый файла данных при помощи среды Borland C. Для этого необходимо выбрать подпункт меню File\New, после чего появится окно, в котором необходимо ввести данные (текст) с клавиатуры, приведенные выше:

Иванов мастер 34

Петров инженер 28

...

Для записи набранного содержимого в созданный файл необходимо выбрать подпункт меню File\Save as, появится окно, в котором необходимо указать путь к файлу (выбрать диск и каталог и указать имя файла 1.dat).

3. Определение структуры программы и выделение ее основных частей (функций).

Необходимо определить структуру программы, т.е. определить набор функций, из которых она будет состоять и как они будут взаимодействовать.

В программе необходимо осуществлять добавления персонала в список персонала фирмы, поэтому на основе базовой структуры Men, описывающей информацию об объекте (человеке), создадим массив Records объектов типа Men. Количество определенных объектов (с которыми можно работать) находится в переменной countRecord.

Пусть в программе требуется реализовать следующие основные функции (функционально и логически законченные подпрограммы):

а) считывание сохраненных ранее данные (LoadFromFile);

б) добавление в массив новой записи (AddRecord);

в) распечатку на экране таблицы (PrintTable);

г) сохранение данных для последующего использования (SavetoFile).

Чтобы получить доступ к структуре Men, массива Records и переменной countRecord из функций программы, объявим их как глобальные объекты вне функций.

В теле основной функции программы реализуется алгоритм, определяющий логику решения задачи и вызов функций в зависимости от введенного значения символа.

4. Непосредственное кодирование функций на языке С и создание программы

Непосредственное кодирование функций LoadFromFile, AddRecord, PrintTable, SavetoFile на языке С выполняется при помощи текстового редактора Borland C++.

Кодирование начинается с разработки алгоритмов указанных функций и функции main, в которой реализуется алгоритм управления программой (в данном случае через меню). В зависимости от используемых стандартных функций, к тексту программы подключается файлы-библиотеки с описанием прототипов (с описанием стандартных функций) при помощи оператора #include.

В программе // - это признак строчного комментария в Borland C++.

Окончательный вариант программы

```
#include <dos.h>            //  библиотека, содержащая функцию delay
#include <stdio.h>          // библиотека, содержащая функции ввода-вывода
#include <conio.h>          // библиотека, содержащая функцию clrscr
struct Man{                 // базовая структура содержит:

   char Name[12];                  //   поле, содержащее фамилию
   char Profession[20];            //   поле, содержащее  профессию
   int Age;                        //    поле, содержащее возраст
};
Man Records[20];            // массив элементов базовой структуры Man
                            // (максимально может быть 20 записей)
int countRecord;            // количество определенных элементов в массиве Records

void LoadFromFile(){        //  функция считывания информации из файла
   FILE *in;            // объявление логического имени файла,
                        // представляющего собой указатель на экземпляр структуры FILE,
                        // описанный в библиотеке stdio.h
   countRecord=0; // обнуление количества считанных
                  // элементов массива Records
   //  конструкция if ((out = fopen(...))= = NULL)while(      !=EOF) позволяет
   // осуществлять на возможность открытия файла для чтения
   if ((in = fopen("c:\\work\\1.DAT", "rt"))== NULL){ // открытия файла
        printf("Cannot open input file.\n");
        return;                                  //  выход в случае неудачи открыть файл
   }
   while (fscanf(in, "%s", Records[countRecord].Name)!=EOF){ // считывание поля Name и
                                            // проверяет не является ли
                                            // считанная информация
```

```
                                        //  признаком конца файла
        if(!fscanf(in, "%s", Records[countRecord].Profession)) // считывание поля Profession
         break;
         if(!fscanf(in, "%d", &Records[countRecord].Age)) // считывание поля Age
         break;
        countRecord++;  // увеличение количества введенных записей в массив на 1
   }
   fclose(in);
}
void SavetoFile(){          //  функция записи информации в файл
   FILE *out; // объявление логического имени файла,
              // представляющего собой указатель на экземпляр структуры FILE,
              // описанный в библиотеке stdio.h
   if ((out = fopen("c:\\work\\1.DAT", "wt"))== NULL){ // открытие файла и проверка на
        printf("Cannot open input file.\n");  //  УСПЕШНОСТЬ ЗАВЕРШЕНИЯ ОПЕРАЦИИ
        return ;                              //  выход в случае неудачи открыть файл
   }
   for(int i=1;i<=countRecord;i++){    // цикл записи информации в файл
                                       // из массива элементов
        fprintf(out, "%s ", Records[i-1].Name);        // запись поля Name в файл
        fprintf(out, "%s ", Records[i-1].Profession); // запись поля Profession в файл
        fprintf(out, "%d\n", Records[i-1].Age);       // запись поля Age в файл
   }
   fclose(out);
}
void PrintTable(){      // функция вывода таблицы на экран
   // создание шапки таблицы
   printf("\n                          ");
   printf("                    Table 1");
   printf("\n*************************************");
   printf("*************************************");
   printf("\n      Name          Profession   Age\n");
   printf("\n*************************************");
   printf("*************************************\n");
   // цикл формирования строк таблицы
   for(int i=1;i<=countRecord;i++){
        printf("%12s ",Records[i-1].Name);
        printf("%20s ",Records[i-1].Profession);
        printf("%5d\n",Records[i-1].Age);
   }
```

```
  // нижняя часть рамка таблицы
   printf("****************************************");
   printf("****************************************\n");
   delay(2000);  // задержка выполнения программы на 2 секунды
}
void AddRecord(){    // функция добавления нового элемента в массив
   printf("input Name: ");                          // ввод имени или фамилии
   scanf("%s",Records[countRecord].Name);
   printf("input Profession: ");                    // ввод адреса
   scanf("%s",Records[countRecord].Profession);
   printf("input Age: ");                           // ввод возраста
   scanf("%d",&Records[countRecord].Age);
   countRecord++;
}
void main(void){
   char ch;
   do{
     clrscr();
     // вывод меню на экран
     printf("L - загрузить данные из файла в массив\n");
     printf("A - добавить сотрудника в массив\n");
     printf("P - распечатать таблицу\n");
     printf("S - запись данных из массива в файл\n");
     printf("V - выход из программы\n");
     // приглашение для ввода и ввод управляющего символа
     printf("Выберите операцию:");
     scanf("%c",&ch);
     // обработка результатов выбора элемента меню
     switch(ch){
        case 'L': LoadFromFile();   // вызов функции считывания информации из файла
                  break;
        case 'A': AddRecord ();       // вызов функция добавления
                                      // нового элемента в массив
                  break;
        case 'P': PrintTable();       // вызов функция вывода таблицы на экран
                  break;
        case 'S': SavetoFile();       // вызов функция записи информации в файл
                  break;
     }
    }while(ch!='V');
```

}

5. Компиляция и отладка программы

На этапе компиляции программы осуществляется проверка правильности написания текста программы. Сообщения об ошибках выводятся в окне внизу в виде списка Error (Warning – это предупреждение о возможной некорректности текста, не является ошибкой).

Результаты выполнения программы можно просмотреть на экране компьютера одновременным нажатием на клавиши Alt+F5. Для возвращения в режим редактирования программы необходимо нажать ту же комбинацию клавиш.

Приложение 5

Работа с (IDE) Borland C++.

Интегрированная среда программирования (IDE) Borland C++.

Интегрированная среда (IDE) – это программа, имеющая встроенный редактор текстов, подсистему работы с файлами, систему помощи, встроенный отладчик, подсистему управления компиляцией и редактированием связей, а также компилятор и редактор связей. Другими словами, IDE дает возможность получить EXE-файл, не используя другие программы. IDE запускается файлом BC.EXE.

После запуска на исполнение файла запуска IDE (файл BC.EXE) на экране отображается основное окно IDE (Рис 1).

Верхняя строка окна – это главное меню. Опции меню позволяют обратиться к подменю и выбрать соответствующую команду.

Нижняя строка экрана отведена под строку состояния, где выделены назначения «горячих» клавиш, воспринимаемых на данном этапе работы.

Выбрать любую из команд меню можно одним из трех способов:

1)нажать клавишу F10 и с помощью клавиш со стрелками выбрать необходимую команду;

2)установить курсор мыши на любое ключевое слово меню и нажать левую кнопку мыши;

3)использовать «горячие» клавиши (метод скорейшего вызова команды). Одновременное нажатие клавиши Alt и «горячей» (клавиша подсвеченная другим цветом).

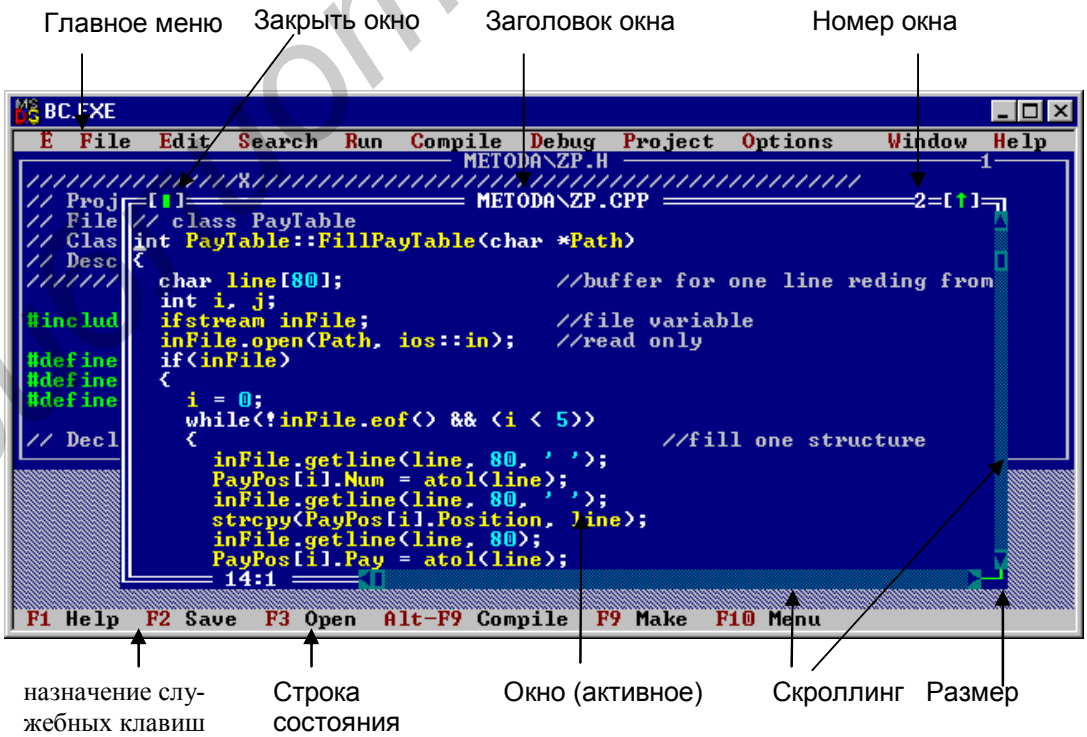


Рис. 1. Основное окно Borland C++ с двумя открытыми окнам

Окно – это ограниченная рамкой область экрана. Его можно открыть, переместить, покрыть другими окнами, изменить размеры, закрыть. Активное окно (то, которое воспринимает нажатия клавиш) обозначается двойной линией. Активное окно изображается всегда поверх других окон. В любой момент только одно окно может быть активным. 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 раскрытия 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)при помощи «горячих» клавиш.

Закрыть можно только активное окно. Для этого либо выбирается в меню Windows команда Close, либо нажимается «горячая» клавиша Alt-F3, либо мышью выбирается поле [•] на окне.

Переключение между окнами выполняется так. Выбирается команд List… из меню Window или нажымается «горячая» клавиша Alt-0. Открывается окно диалога, в котором можно выбрать любое из открытых и ранее закрытых окон. Если на экране отображена хотя бы небольшая часть необходимого окна, достаточно в эту область установить мышь и нажать ее левую кнопку, чтобы окно стало активным. Циклический просмотр окон возможен при помощи клавиши F6.

Активное окно может быть раскрыто на весь экран либо выбором Window-Zoom, либо нажатием клавиши F5, либо выбором мышью поля [↑].

Для просмотра результатов выполнения программы, если вывод выполняется в текстовом режиме, используется переключение в окно вывода (Window - Output). Для просмотра результатов как в текстовом, так и графическом режимах, следует активизировать окно экрана пользователя (Window – User screen) или воспользоваться . одновременным нажатием клавиши – Alt-F5. Возврат в среду происходит при нажатии любой клавиши.

Переключение в режим редактирования выполняется автоматически при выборе команды New в меню File или при открытии файла. Для возврата из меню в режим редактирования достаточно нажать клавишу Esc.

Команды вставки и удаления (под блоком понимается выделенное подсветкой подмножество символов):

Ins – режим вставки/замены;

Del – удалить символ в позиции курсора;

Backspace – удалить символ слева от курсора;

Ctrl-Y – удалить строку;

Ctrl-N – вставить строку.

Команды работы с блоками:

Shift+клавиши со стрелками – выделение блока текста;

Ctrl-Ins – копировать блок в буффер обмена;

Shift-Ins – копировать блок из буффера обмена в текущую позицию курсора;

Ctrl-Del – удалить блок.

Shift -Del – вырезать блок в буфер обмена.

Этапы создания программы в инструментальных средах фирмы Borland.

Основные этапы создания программы в IDE Borland C++:

1)настройка опций среды программирования;

2)набор исходного текста программы;

3)компиляция программы;

4)компоновка программы;

5)отладка программы;

6)запуск программы на исполнение.

Система программирования Borland C++ включает:

1)интегрированную среду программирования (Integrated Development Environment - IDE);

2)компилятор исходного текста программы;

3)редактор связей (компоновщик);

4)библиотеки заголовочных файлов;

5)библиотеки функций;

6)программы-утилиты.

Задание опций интегрированной среды

Первым шагом при работе с IDE является настройка нужных опций (дополнительных параметров). Все опции имеют значения по умолчанию. Рассмотрим основные опции, настраиваемые с помощью команд меню Options.

Для того чтобы начать работу с IDE, прежде всего требуется задать директории, используемые текстовым редактором, компилятором и компоновщиком (см. рис. 2). Для этого используется команда Options\Directories (мы будем использовать формат записи Меню\Меню\…\Команда для экономии места). Поле ввода Include Directories используется для задания директориев заголовочных файлов. В поле ввода разрешается указывать несколько директориев, разделяемых символом “;”.
Поле ввода Library Directories задает директории, содержащие объектный файл загрузчика (C0?.OBJ, где ? – это буква M, S, H, T, L, C в зависимости от используемой модели памяти) и файлы библиотек функций (.LIB). Поле ввода Output Directory задает директорий, в котором помещаются файлы с расширениями .OBJ, .EXE, .MAP. Если в поле – пустая строка, используется текущий директорий.

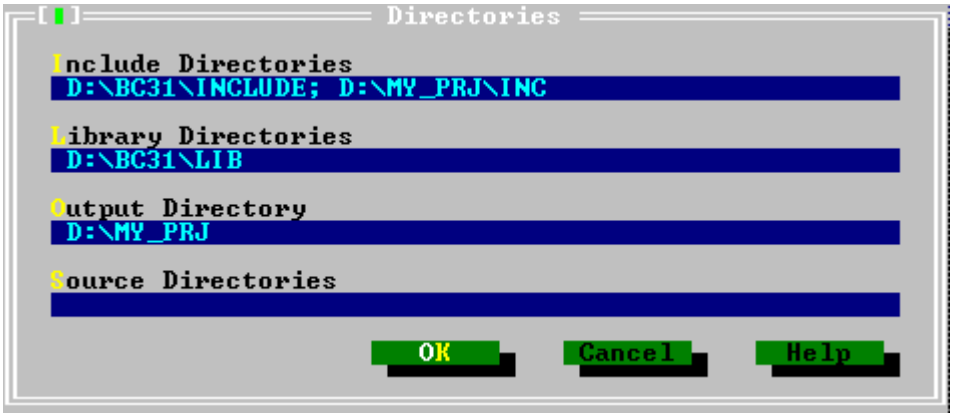


Рис. 2. Окно диалога для установки директориев

При выборе строки Options\Compiler открывается еще одно меню для настройки опций компилятора. Наиболее важные опции задаются при выборе команды Code generation. Опция считается выбранной, если она помечена символом (•), и включенной, если она помечена символом [x]. Самым важным пунктом в окне Code generation является выбор модели памяти. Для большинства программ, разрабатываемых для ОС MS-DOS, нужно выбрать SMALL модель памяти.

Набор текста программы

Следующим шагом является ввод программы с использованием текстового редактора и сохранение исходного текста программы в файле.

Набор текста программы можно выполнить встроенным или любым другим текстовым редактором. По традиции файлы, содержащие исходные тексты программ на языке Си, имеют расширение имени файла .C, а на языке Си++ - .CPP.

Не следует начинать компиляцию, компоновку или запуск программы без сохранения сделанного набора! Запущенная на выполнение программа может вызвать «зависание» компьютера, и сделанный набор будет потерян.

К программам-утилитам относят ассемблер, препроцессор, отладчик, программу профилирования и многие другие полезные программы.

Компиляция, редактирование связей, запуск программы на выполнение

Borland C++ включает богатейшие библиотеки функций для управлениями файлами, выполнения ввода-вывода и многих других действий. Прототипы (заголовки функций с описанием типов формальных параметров и типа возвращаемого функцией значения), символические константы и другие макро, связанные с библиотечными функциями, объединяют в заголовочные файлы, которые по традиции имеют расширение .H. Необходимые для компиляции файлы включаются в текст программы при помощи препроцессорной директивы #include. При запуске на компиляцию текст программы сначала обрабатывается препроцессором, который обрабатывает только «свои» директивы (в частности, вместо директивы «#include имя_файла» встраевается

из библиотеки INCLUDE файл, имя которого задано в директиве), а затем текст программы передается непосредственно на обработку компилятору.

Компиляция исходного текста программы инициируется либо через команду Compile\Compile to OBJ, либо нажатием «горячей» клавиши Alt-F9. Команда Make EXE file также запускает программу на компиляцию и при отсутствии синтаксических ошибок автоматически запускает компоновщик для получения .EXE-файла. Еще одна возможность для запуска программы на компиляцию – команда Run\Run (Ctrl-F9). После успешной компиляции и компоновки запускается полученный .EXE-файл на выполнение.

Все сообщения об ошибках и предупреждения IDE помещает в окно по имени Message. Это окно активно после завершения компиляции. Если в программе обнаружены ошибки, включаются средства трассировки ошибок, 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. Нажатие клавиши F6 (переход или активизация следующего окна) вновь делает активным окно Message.

Многофайловая компиляция

При модульном программировании не обойтись без многофайловай компиляции. 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 функций. Имена этих файлов записываются в специальный файл – файл проекта, из которого IDE узнает, какие из текстовых файлов следует объединять в исполняемый (.EXE) файл.

Все необходимые для работы с файлами проектов команды включены в меню Project.

Для организации файла проекта необходимо открыть файл проекта. Для этого выполняется команда Project\Open Project… IDE активизирует специальное окно Project в нижней части экрана и открывает окно диалога, позволяющее загрузить нужный файл проекта или создать новый с заданным именем.

Если создан новый файл проекта, окно Project первоначально будет пустым. Включение файлов в проект и их удаление выполняются либо через команды Project\Add item… и Project\Delete item, либо нажатием клавиш Ins и Del, в случае если курсор размещен в окне Project. При добавлении файлов в проект открывается окно диалога, позволяющее выбрать нужный файл.

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 клавиша ENTER.

При работе со средой Borland C++ рекомендуется использовать проект, даже если программа состоит из одного файла.

Отладка программы.

В процессе отладки вы можете:

1)осуществлять пошаговое выполнение программы. После прохода каждой ее строки будет производиться приостановка, позволяющая проанализировать промежуточные результаты;

2)проверять значение и местоположение (адрес) некоторой переменной в ходе выполнения программы;

3)просмотреть последовательность вызова функций в программе.

Существует два режима пошагового выполнения программы:

1)трассировка с заходом в тело функции, при встрече ее вызова в тексте программы (F7);

2)пошаговое выполнение функции (как обычной команды без захода в тело функции), при встрече ее вызова в тексте программы (F8).

Команда Run\Trace into (F7) запускает программу на отладку. Интегрированная среда высвечивает строку программы, содержащую точку входа main(). После этого нажатием клавиши F7 вызывается выполнение кода, соответствующего одной строке текста программы. Если в строке записана ссылка на функцию, начинается трассировка по тексту тела функции. При необходимости выполнения строки функции за один шаг, используется клавиша F8 (команда Run\Step over).

Для ускорения процесса отладки используется команда Run\Go to cursor (F4). Программа выполняется до строки, в которой в данный момент располагается текстовый курсор. Можно также задать режим выполнения до точки останова (через опцию подменю “Debug\Toggle breakpoint” или одновременным нажатием клавиш Ctrl и F8, в дальнейшем будем использовать запись “Ctrl+F4”). При этом строка в точке останова подсвечивается обычно красным фоном. Снять установку точку останова можно повторным выполнением описанной команды, размещая курсор на подсвеченной строке останова.

Для наблюдения за изменением значений переменных в ходе выполнения программы используется подменю Debug\Watches\Add watch или “Ctrl+F7”. В появившемся окне Add Watch (вызов окна Add Watch можно также получить если нажать клавишу Ins, предварительно сделав активным окно Watch) необходимо ввести имя переменной, значение которой необходимо просмотреть и нажать Ввод. Указанная переменная размещается в окне Watch, создаваемом в нижней части экрана, и, в процессе отладки, через это окно можно наблюдать за изменением размещенных в нем переменных. Удалить переменную из окна Watch можно при помощи клавиши “Del”, предварительно выделив ее подсветкой.

Используя опцию меню Evaluate/modify или “Ctrl+F4”, можно изменить значение

переменной в процессе выполнения отладки, чтобы протестировать алгоритм с новым заданным значением. Окно этой опции “Evaluate and Modify” можно использовать и в качестве калькулятора, если записать выражения с переменными в строке “Expression” и нажать клавишу “Evaluate’ для получения результата в строке Result.

Приложение 6

Использование глобальных переменных, объявленных вне файла.

Ниже приведена простая программа на языке С, состоящая из двух файлов (модулей) - first.c (главный файл) и second.c:

```
// начало первого файла first.c
#include <stdio.h>                  // подключение библиотеки stdio.h
extern int a, b;                    // объявление переменных объявленных в
                                    // другом файле (в файле second.c)
void main(void)                     // основная функция main
{
   printf( "a = %d  b = %d\n", a, b );   // вывод значений переменных a,b
}
```

Во втором файле second.c должно присутствовать объявление и инициализация переменных, например, в виде:

```
int a = 2, b = 5;                   // объявление переменных a,b
```

Две глобальные переменные a и b типа int описаны вне функции в файле second.c. Следовательно, их имена являются глобальными в файле, т.е. они видимы для загрузчика языка С. Поскольку в файле first.c присутствует описание extern для этих переменных типа int, то компилятор разрешает использовать их в функциях first.c, например в main(). Объявление extern int a,b можно поместить как внутри, так и вне функций (например функции main()). При объявлении вне функции переменные a и b будут видимы во всем модуле first.c. Если оператор extern int a,b помещен внутри функции main(), то переменные будут видимы только внутри этой функции.

Если перед описанием int a=2, b=5 в файле second.c поместить обозначение класса памяти static, то указанные переменные скрываются от загрузчика. Такие переменные известны только в пределах файла, в котором они объявлены. Хотя программа в файле first.c будет успешно оттранслирована (компилятор учтет описание extern), загрузчик не сможет уточнить адреса для внешних ссылок и выдаст соответствующее сообщение об ошибке.

Св. план 1999, поз. 93

Учебное издание

Авторы: Бахирев Андрей Владимирович,
Живицкая Елена Николаевна,
Комличенко Виталий Николаевич,
Соколов Сергей Александрович,
Синицын Александр Григорьевич.

МЕТОДИЧЕСКОЕ ПОСОБИЕ И УЧЕБНЫЕ МАТЕРИАЛЫ по курсу «Основы информатики и вычислительной техники» для студентов экономических специальностей заочной формы обучения в 2-х частях

Часть 1

Редактор Е.Н. Батурчик

Подписано в печать Формат 60х84 1/16.
Бумага Печать офсетная. Усл. печ. л.
Уч.-изд. л. 4,6. Тираж 120 экз. Заказ

Белорусский государственный университет информатики и радиоэлектроники
Отпечатано в БГУИР. Лицензия ЛП №156. 220027, Минск, П. Бровки, 6