ОБРАБОТКА ИНФОРМАЦИИ В СОВРЕМЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Основы алгоритмизации

Работа в интегрированной среде TURBO PASCAL

г.Днепропетровск

СОДЕРЖАНИЕ

1. АЛГОРИТМ И ЕГО ОСНОВНЫЕ СВОЙСТВА

1.1. Этапы решения задачи на ЭВМ

Решение любой задачи на ЭВМ состоит из нескольких этапов, среди которых основными являются:

1) **постановка задачи –** точная формулировка условий и целей ее решения;
2) **формализация** (*математическая* постановка задачи) – построение математической модели рассматриваемого в задаче явления;
3) **выбор** (или разработка) **метода решения** – построение ряда формул и формулировка правил, определяющих связи между этими формулами;
4) **разработка алгоритма** – (алгоритмизация) – разложение вычислительного процесса на возможные составные части, установление порядка их следования, описание содержания каждой такой части в той или иной форме;
5) **составление программы** (программирование) – изложение разработанного алгоритма задачи на языке, который будет понятен ЭВМ;
6) **отладка программы –** исправление ошибок, допущенных на предыдущих этапах;
7) **вычисление и обработка результатов –** непосредственное решение задачи на компьютере и оценка реальности результатов.

1.2. Понятие алгоритма

Алгоритм – **это точное предписание исполнителю выполнить конечную последовательность команд, приводящую от исходных данных к искомому результату**.

Термин «алгоритм» имеет древнее происхождение. Являясь латинизированной транскрипцией имени великого среднеазиатского ученого IX века Мухаммеда аль - Хорезма.

Примерами алгоритмов могут быть изучающиеся в школе правила сложения, умножения чисел, правила построения геометрических фигур, грамматические правила правописания слов и предложений.

1.3. Свойства алгоритма

- *Дискретность*. Процесс решения задачи должен быть разбит на последовательность отдельных (дискретных)шагов. Только выполнив одну команду, исполнитель сможет приступить к выполнению следующей.
- *Точность*(определенность). Каждая команда алгоритма должна определять однозначное действие исполнителя. Это требование называется точностью алгоритма.
- *Понятность*. Алгоритм, составленный для конкретного исполнителя, должен включать только те команды, которые входят в его систему команд.
- *Конечность*(результативность). Исполнение алгоритма должно завершиться за конечное число шагов.
- *Массовость*. Алгоритм должен обеспечивать решение всего класса задач данного типа.

Алгоритм считается правильным, если его выполнение дает правильный результат. Соответственно алгоритм содержит ошибки, если можно указать такие допустимые исходные данные или условия, при которых выполнение алгоритма либо не завершится вообще, либо не будет получено никаких результатов, либо полученные результаты окажутся неправильными.

1.4. Классы алгоритмов

Выделяют три крупных класса алгоритмов:

- вычислительные алгоритмы, работающие со сравнительно простыми видами данных, такими как числа и матрицы, хотя сам процесс вычисления может быть долгим и сложным;

- информационные алгоритмы, представляющие собой набор сравнительно простых процедур, работающих с большими объемами информации (алгоритмы баз данных);

- управляющие алгоритмы, генерирующие различные управляющие воздействия на основе данных, полученных от внешних процессов, которыми алгоритмы управляют.

2. ОБЩИЕ ПРАВИЛА ЗАПИСИ АЛГОРИТМОВ

2.1. Стандартизация описания алгоритмов

Существует несколько способов записи алгоритмов

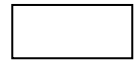
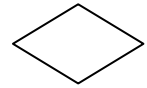
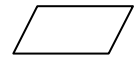
1) Словесная запись алгоритмов – описание алгоритма средствами обычного языка, но с тщательно отобранным набором слов и фраз, не допускающим повторений, синонимов, двусмысленностей, лишних слов; кроме того, при таком описании существуют определенные соглашения о форме записи, порядке выполнения действий, допускается использование математических символов.

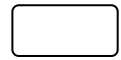
2) Псевдокод – метод применения естественного языка для описания алгоритма, но с конструкциями, близкими к конструкциям языков программирования, что дает возможность создать машинно-независимое описание алгоритма (т.н. формальные алгоритмические языки – например, алгоритмический язык академика А.П.Ершова) .

3) Блок-схемы алгоритмов – это графическое изображение алгоритма с помощью отдельных блоков, обозначающих то или иное действие.

Блок – графический символ, соответствующий одному шагу алгоритма. Внутри блока дается описание соответствующего действия.

2.2. Основные блоки алгоритмических схем

№	Название символа	Символ	Отображаемая функция
1.	Блок вычислений (процесс)		Вычислительное действие (или последовательность вычислительных действий)
2.	Логический блок		Выбор направления выполнения алгоритма в зависимости от некоторых условий(условия)
3.	Блоки ввода-вывода		Общее обозначение ввода или вывода данных (вне зависимости от физического носителя)
	Блоки вывода		Вывод данных, носителем которых служит документ (печатающее устройство)

№	Название символа	Символ	Отображаемая функция
4.	Начало – конец (вход – выход)		Начало или конец программы, остановка, вход или выход в подпрограммах
5.	Подпрограмма		Вычисление по стандартной подпрограмме или подпрограмме пользователя
6.	Блок модификации (заголовок цикла)		Выполнение действий, изменяющих пункты алгоритма
7.	Соединитель		Указание связи между прерванными линиями потока информации в пределах одной страницы
8.	Межстраничный соединитель		Указание связи между частями схемы, расположенными на разных листах

3. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

3.1. Развитие языков программирования

Совокупность средств и правил представления алгоритма в виде, пригодном для выполнения вычислительной машиной, называется *языком программирования*, а каждый алгоритм, записанный на некотором языке программирования, называется *программой*.

Первые языки программирования были очень примитивными и мало чем отличались от формализованных упорядоченных последовательностей единиц и нулей, понятных компьютеру. Использование таких языков было крайне неудобно с точки зрения программиста, так как он должен был знать числовые коды всех машинных команд, должен был сам распределять память под команды программы и данные.

Для того чтобы облегчить общение человека с ЭВМ были созданы языки программирования типа **Ассемблер**. Переменные величины стали изображаться символическими именами. Числовые коды операций заменились на мнемонические обозначения, которые легче запомнить. Язык программирования приблизился к человеческому языку, и отдалился от языка машинных команд.

Один из первых языков программирования – **Фортран** (Formula Translation) был создан в середине 50-х годов, который и в наши дни остается одним из самых распространенных. Он используется для инженерных и научных расчетов, для решения задач физики и др. Для решения экономических задач был создан язык программирования - **Кобол**. Расширение областей применения ЭВМ

влечет за собой создание языков, ориентированных на новые сферы применения: **Снобол** – алгоритмический язык для обработки текстовой информации, **Лисп** - алгоритмический язык для обработки символов. **Лисп** находит широкое применение в исследованиях по созданию искусственного интеллекта.

В 1968 г. был объявлен конкурс на лучший язык программирования для обучения студентов. Победителем стал **Алгол-68**, но широкого распространения не получил. Для этого конкурса **Николаус Вирт** создал язык **Паскаль**, достаточно простой, удобный, с наличием мощных средств структурирования данных. Хотя **Паскаль** был разработан как язык для обучения программированию, он впоследствии получил широкое развитие и в настоящее время считается одним из самых используемых языков. Для обучения младших школьников **Самуэлем Пайпертом** был разработан язык **Лого**. Он отличается простотой и богатыми возможностями.

Широкое распространение в школах в качестве обучающего языка получил язык Бейсик. Спустя много лет после изобретения Бейсика, он и сегодня самый простой для освоения из десятков языков общецелевого программирования.

Необходимость разработки больших программ, управляющих работой ЭВМ, потребовала создания специального языка программирования **СИ** в начале 70-х г. Он является одним из универсальных языков программирования. В отличии от Паскаля, в нем заложены возможности непосредственного обращения к некоторым машинным командам и к определенным участкам памяти компьютера. **СИ** широко используется как инструментальный язык для разработки операционных систем, трансляторов, баз данных и других системных и прикладных программ.

Появление функционального программирования привело к созданию языка **Пролог.** Этот язык программирования разрабатывался для задач анализа и понимания естественных языков на основе языка формальной логики и методов автоматического доказательства теорем.

В 80-х г. 20 века был создан язык **Ада**. Этот язык в дополнение к классическим свойствам, обеспечивает программирование задач реального времени и моделирования параллельного решения задач.

3.2. Классификации языков программирования

Существуют различные классификации языков программирования. По наиболее распространенной классификации все языки программирования делят на языки

- **низкого** уровня
- **высокого** уровня
- **сверхвысокого** уровня.

В группу языков **низкого уровня** входят машинные языки и языки символического кодирования: (Автокод, Ассемблер). Операторы этого языка – это те же машинные команды, но записанные мнемоническими кодами. Все языки низкого уровня ориентированы на определенный тип компьютера, т. е. являются машинно-зависимыми.

Более многочисленную группу составляют языки программирования **высокого уровня**. Это Фортран, Алгол, Кобол, Паскаль, Бейсик, СИ, Пролог и т.д. Эти языки машинно-независимы, т.к. они ориентированы на систему операндов, характерных для записи определенного класса алгоритмов. Однако программы, написанные на языках высокого уровня, занимают больше памяти и медленнее выполняются, чем программы на машинных языках.

К языкам **сверхвысокого уровня** можно отнести лишь Алгол-68 и APL. Повышение уровня этих языков произошло за счет введения сверхмощных операций и операторов.

Другая классификация делит языки на **вычислительные** и языки **символьной обработки.** К первому типу относят Фортран, Паскаль, Алгол, Бейсик, Си, ко второму типу - Лисп, Пролог, Снобол и др.

В современной информатике можно выделить два основных направления развития языков программирования: **процедурное** и **непроцедурное**. Среди процедурных языков выделяют в свою очередь **структурные** и **операционные языки.** Структурные языки Паскаль, Си, Ада, ПЛ/1. Среди операционных известны Фортран, Бейсик.

Можно выделить еще один класс языков программирования - **объектно-ориентированные языки высокого уровня,** которые благодаря богатому пользовательскому интерфейсу, предлагают человеку решить задачу в удобной для него форме. Примером такого языка может служить язык программирования визуального общения Object Pascal.

Языки описания сценариев, такие как Perl, Python, Rexx, Tcl и языки оболочек UNIX, предполагают стиль программирования, весьма отличный от характерного для языков системного уровня. Они предназначаются не для написания приложения с нуля, а для комбинирования компонентов, набор которых создается заранее при помощи других языков. Развитие и рост популярности Internet способствовали распространению языков описания сценариев, а среди разработчиков Web-страниц популярен JavaScript.

4. БАЗОВЫЕ АЛГОРИТМИЧЕСКИЕ КОНСТРУКЦИИ

4.1. Структурное программирование и базовые конструкции

Структурное программирование – разработка и документирование алгоритмов и программ, обеспечивающая легкость понимания алгоритма, простоту для проверки правильности алгоритма, удобство модификации.

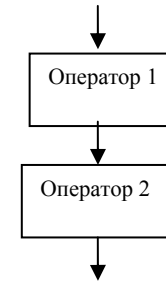
В основу структурного программирования положены следующие достаточно простые положения:

1. Алгоритм и программа должны составляться поэтапно (по шагам).
2. Сложная задача должна разбиваться на достаточно простые, легко воспринимаемые части, каждая из которых имеет только один вход или один выход.
3. Логика алгоритма и программы должна опираться на минимальное число достаточно простых **базовых управляющих структур**, обладающих функциональной полнотой

В процессе алгоритмизации задачи исходный алгоритм разбивают на отдельные *частные алгоритмы* (шаги). Частные алгоритмы могут быть линейными, разветвляющимися и циклическими.

4.2. Линейные алгоритмы

Линейный алгоритм – набор команд, выполняемых последовательно во времени один за другим.



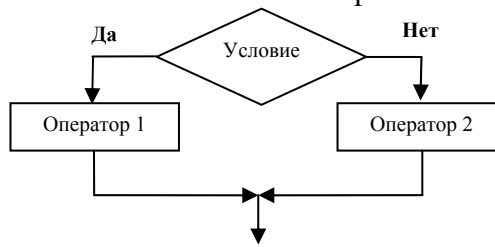
Базовая конструкция линейного алгоритма в блок-схеме имеет вид последовательно расположенных блоков, без ветвлений и возвратных потоков.

Конструкция широко используется при составлении алгоритмов вычислений при заранее определенных значениях переменных.

4.3. Алгоритмы с разветвлениями

Разветвляющийся алгоритм – алгоритм, содержащий хотя бы одно условие, в результате проверки которого обеспечивается переход на один из двух возможных шагов. По этой команде исполнитель выбирает один из двух путей исполнения алгоритма с непременным выходом на общее продолжение. Выбор происходит по какому-либо *условию*.

Условие – это высказывание, о котором можно сказать истинно оно или ложно. Условие может быть простым или составным.



4.4 Циклические алгоритмы

Алгоритмы, содержащие команды повторения называют **циклическими.** Команды повторения составляют *цикл.*

Цикл программы – это последовательность команд (*серия, тело цикла*), которая может выполняться *многократно* (для новых исходных данных) до удовлетворения некоторого *условия.*

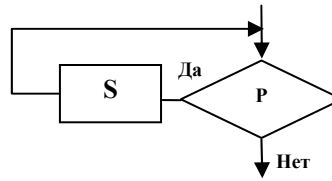
Различают три вида циклов: *цикл - пока, цикл-до* и цикл с пошаговым изменением аргумента.

Любой цикл состоит из нескольких этапов:

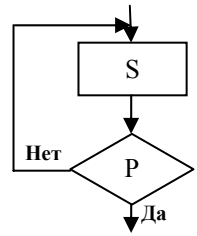
1. Подготовка цикла, в которую входят начальные присвоения.
2. Тело цикла – команды присвоения цикла.
3. Условие – обязательная часть циклов «До» и «Пока».

К циклическим алгоритмам сводится большинство методов вычисления и перебора вариантов.

Цикл – пока (с предусловием). В него входят в качестве базовых следующие структуры: блок проверки условия P и блок и блок S , называемый телом цикла.

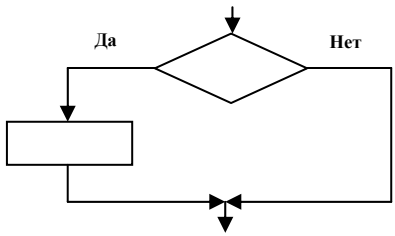


Цикл – до (с постусловием). Возможен другой случай, когда тело цикла S выполняется по крайней мере один раз и будет повторяться до тех пор, пока не станет истинным условие P.

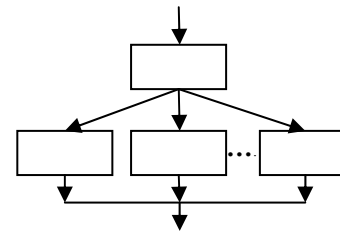


Итак, **цикл – до** завершается, когда условие P становится истинным, а **цикл- пока** – когда P становится ложным.

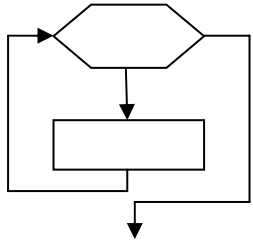
В языке Паскаль, в котором наиболее полно нашли свое отражение идеи структурного программирования, целесообразно при проектировании алгоритмов использовать четыре элементарные структуры. Каждая из этих структур имеет один вход и один выход.



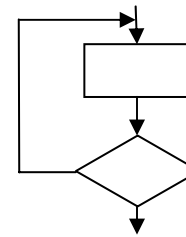
Сокращенная запись разветвления



Структура варианта



Структура цикла с параметром



Структура цикла с постусловием

В языке Паскаль имеются средства (операторы), позволяющие непосредственно реализовать в программе любую из этих структур, поэтому правильное использование типовых структур в процессе разработки алгоритма обеспечивает упрощение следующих этапов решения задачи на ЭВМ.

Эффективным методом построения алгоритмов является *метод пошаговой детализации* (последовательного построения).

При этом сложная задача разбивается на ряд более простых. Для каждой подзадачи составляется свой алгоритм. При этом каждый из дополнительных алгоритмов должен как решать свою подзадачу, так и стыковаться с остальными дополнительными алгоритмами в основном. Эти подзадачи могут, в свою очередь, потребовать разбиения на еще более простые задачи и.т.д.

Метод пошаговой детализации лежит в основе так называемого структурного программирования.

5. РАБОТА В ИНТЕГРИРОВАННОЙ СРЕДЕ TURBO PASCAL

Обычное имя каталога с Турбо Паскалем – TP. Интегрированная среда, редактор, компилятор, компоновщик и отладчик содержатся в файле TURBO.EXE. На экране появляется окно, состоящее из нескольких частей.

Строка меню предоставляет доступ к командам интегрированной среды. Активизировать строку меню можно, нажав клавишу F10. Можно воспользоваться для этого и мышью, указав на меню и нажав левую кнопку.

Имеется еще один способ быстрой активизации пункта меню.

Для этого следует нажать клавишу <Alt> и выделенную букву в названии пункта.

Если за командой меню следует знак многоточия (…), то выбор этой команды приведет к выводу диалогового окна. Если за командой следует значок >, то эта команда вызывает вложенное меню. Все прочие команды непосредственно связаны с выполнением каких-то действий. Запуск команды производится нажатием клавиши Enter.

Прервать выполняемое действие можно нажатием клавиши Esc, а прервать выполнение программы – одновременным нажатием клавиш Ctrl+Break.

Некоторые команды меню могут быть недоступными (их названия выводятся серым цветом). Это происходит в том случае, когда использование команды не имеет смысла.

Строка статуса. Она содержит напоминание о назначении основных комбинаций клавиш.

Для сохранения настроек интегрированной среды используется файл конфигурации TURBO.TP.

Основные опции интегрированной среды

Команда **Options** > **Compiler** позволяет установить опции компилятора.

Группа **Memory Sizes** используется для определения размера стека, а также минимального и максимального количества динамически распределяемой памяти.

Группа **Linker** определяет опции компоновщика.

Группа **Debugger** определяет вид отладочной информации, включаемой в исполняемый файл.

Расположение необходимых файлов указывается в окне, вызываемой командой **Directories**.

В первом поле перечисляются каталоги, в которые помещаются результаты компиляции – исполняемые файлы и скомпилированные модули, во втором поле – включаемые файлы, а в третьем и четвертом полях – каталоги с модулями и объектными файлами, порожденными ассемблерными программами.

Команда **Tools** позволяет присоединить к интегрированной среде дополнительные утилиты, такие как утилита поиска *grep*, программа профилирования Turbo Profiler и.т.д.

Команда меню **Environment** используется для настройки интерфейса интегрированной среды.

Установки можно сохранить в *файле* (точнее говоря, в файлах с расширениями .TP и .DSK) или загрузить из ранее сформированного файла, используя команды Save, Save As и Open.

Набор и редактирование исходного текста программы

Набор и редактирование исходного текста программы производится средствами встроенного текстового редактора интегрированной среды.

Если раскрыть меню **Edit**, то можно увидеть перечень функций редактора, доступных через меню. Этот перечень включает (по порядку): отмену предыдущего действия (Undo), восстановление предыдущего действия (Redo), удаление в буфер(Cut), копирование в буфер (Copy), вставку из буфера (Paste) и удаление (Clear). Справа от наименования функций показаны связанные с ними клавиши или комбинации клавиш.

Специальные клавиши редактирования

Клавиши	Команда меню	Функция
Ctrl+Del	Edit>Clear	Удаление выбранного текста в буфер
Ctrl+Ins	Edit>Copy	Копирование выбранного текста в буфер
Shift+Del	Edit>Cut	Перемещение выбранного текста в буфер
Shift+ Ins	Edit>Paste	Запись текста из буфера в активное окно
F2	File>Save	Сохранение файла, находящегося в активном окне редактора
F3	File> Open	Открытие файла

Создание новой программы.

После запуска интегрированной среды на экране должно появиться пустое активное окно редактирования. Если появившееся окно не пустое, то с помощью команды меню File>New следует активизировать окно ввода нового текста. В верхней части окна редактирования появится название, которое среда автоматически присваивает новому файлу, - NONAME00.PAS.

После набора текста программы надо обязательно изменить имя файла, иначе есть опасность потерять его, если он случайно будет замещен другим файлом с таким же стандартным именем.

Рекомендуется также периодически производить сохранение файла с помощью нажатия клавиши F2 (через каждые 10-20 набранных строк), так как всегда имеется вероятность аварийной ситуации в работе компьютера, после которой несохраненный на диске файл будет потерян.

При первой записи файла на диск система предложит задать имя файла, причем расширение . PAS добавляется автоматически.

С редактированием связан еще один пункт меню – Search, который используется для поиска и замены фрагментов текста.

Чтобы выполнить поиск, необходимо указать искомый фрагмент текста («образец» для поиска). Если курсор находится в каком-то слове в окне редактирования, это слово по умолчанию будет об-

разцом для поиска командах Find (поиск) или Find\Replace(поиск и замена).

Работа в меню File

Открыть файл: File> Open (клавиша F3), появляется приглашение ввести имя файла. Расширение .PAS добавляется к имени файла автоматически.

Сохранить файл: File>Save (клавиша F2), или File>Save as – сохранить файл под другим именем, или File>Save all – сохранить все открытые в среде файлы.

Закрыть файл: Window>Close, или Alt+ F3 или щелчком на маленьком прямоугольнике в левом верхнем углу активного окна. Если с момента предыдущей записи на диск выполнялось редактирование файла, последует приглашение сохранить результаты этого редактирования.

Функция DOS Shell позволяет выполнить команды MS-DOS, не выгружая интегрированную среду. Вернуться из режима выполнения команд MS-DOS можно, набрав команду Exit.

В нижней части меню File находится список последних открывавшихся файлов. На любой из строк этого списка можно щелкнуть мышью, чтобы открыть соответствующий файл вновь.

Закрыть интегрированную среду - File>Exit, или Alt+X.

Работа с меню Window.

Меню Window содержат команды, которые позволяют упорядочить открытые окна программ(Tile, Cascade), закрыть активные окна(Close, Close all), изменить размер окна, переместить окно(Size/Move, Zoom).

Команды Next и Previous (циклическое переключение между окнами), которым сопоставлены клавиши F6 и Shift+ F6 .

Команда Refresh display(обновить экран) - при возвращении в интегрированную среду (при работе с графической программой) экран может оказаться нарушенным и пропадает связь с мышью.

Работа со справочной системой

В процессе набора текста программы достаточно часто приходится пользоваться встроенной справочной системой Турбо Паскаля. Справочная система является контекстно-зависимой, то есть выводимая при обращении к ней информация зависит от того, где в момент обращения находится курсор.

При нажатии клавиши **F1** выводится справочная информация, относящаяся к редактированию файла или активизированному пункту меню. Самый быстрый способ вызова справочной системы – использование специальных клавиш или их комбинаций.

Клавиши	**Команда меню**	**Функция**
F1	Help>Contents	Открывает контекстно-ориентированный экран справочной информации
F1+ F1	Help> Help on Help	Вызов справки по справочной системе
Shift+ F1	Help>Index	Вызов оглавления справочной системы
Alt+ F1	Help>Previous Topic	Показать предыдущий экран справочной информации
Ctrl+ F1	Help> Topic Search	Вызов контекстной информации по языку Паскаль.

Выход из среды Турбо Паскаля

Чтобы выйти из интегрированной среды, нажмите клавиши Alt+X. При этом будет предложено сохранить каждый из открытых файлов, если результат их редактирования не был предварительно записан на диск.

Совет:

- Можно использовать клавишу **Esc** для выхода из трудных ситуаций при работе в интегрированной среде.

- Нажатие на правую кнопку мыши выводит в окно редактирования **краткое меню**, содержащее небольшой, но полезный набор функций интегрированной среды. Это же меню вызывается нажатием клавиш Alt+ F10.

Компиляция программы, поиск и устранение ошибок

После завершения набора программы и ее записи на диск можно приступать к компиляции (клавиша **F9** или **Compile> Compile**).

При этом компилятор может обнаружить ошибки в программе. Сообщение об ошибке выводится в верхней части экрана и выделяется красным цветом. Курсор при этом устанавливается в той строке программы, где обнаружена ошибка.

Исправление ошибок сводится к внесению изменений в исходный текст программы. Компилятор, разумеется, не может найти алгоритмические ошибки, но ко всему, что касается синтаксиса, он относится достаточно строго!

Результат успешной компиляции – файл, имя которого совпадает с именем исходного файла, а расширение является стандартным расширением исполняемого файла - .EXE.

Этот файл может размещаться в оперативной памяти (в этом случае он будет потерян при выходе из интегрированной среды) или на диске. Задать размещение исполняемого файла можно с помощью команды **Compile**>**Destination**, которая является переключателем двух возможных значений – **Memory** (исполняемый файл в оперативной памяти) и **Disk**(исполняемый файл на жестком диске). Для переключения между этими вариантами необходимо установить указатель на команду Destination и нажать клавишу **Enter**.

Далее в меню Compile следуют команды, позволяющие задать основной файл проекта, а также команда Information, активизация которой позволяет получить полезную информацию о скомпилированной программе (размер программы, распределение оперативной памяти и.т.д.).

Запуск программы на выполнение, просмотр результатов

После завершения «борьбы» с синтаксическими ошибками можно запускать программу на выполнение (клавиши **Ctrl+ F9** или **Run>Run**).

Программа может благополучно отработать, но могут обнаружиться ошибки времени выполнения или алгоритмические ошибки. Если в программе не предусмотрена приостановка выполнения для просмотра результатов работы, выведенные на экран результаты перекрываются окном интегрированной среды. (Для приостановки программы во время отладки можно использовать оператор READLN без параметров).

Временно убрать это окно и просмотреть результаты работы можно, нажав клавиши **Alt+ F5.** После просмотра нажатие любой клавиши вернет на экран рабочее поле редактора.

Специальные клавиши и комбинации клавиш для запуска и отладки программ.

Клавиши	Команда меню	Функция
Alt+ F9	Compile> Compile	Компиляция активного файла
Ctrl+ F2	Run>Program Reset	Сброс выполняемой программы в исходное состояние
Ctrl+ F4	Debug> Evaluate/Modify	Вычислить выражение или модифицировать значение переменной
Ctrl+ F7	Debug> Add Watch	Добавить выражение для просмотра
Ctrl+ F8	Debug> Toggle Breakpoint	Установить или удалить точку прерывания
Ctrl+ F9	Run> Run	Запустить программу на выполнение
F4	Run> Go to Cursor	Выполнение фрагмента программы от подсвеченной строки до строки, на которой стоит курсор
F7	Run>Trace Into	Выполнение одной строки программы с заходом внутрь процедур
F8	Run>Step Over	Выполнение одной строки программы, минуя вызовы процедур
F9	Compile>Make	Компиляция программы из текущего окна

Поиск ошибок и методы отладки программы

В процессе создания новой программы программисту приходится сталкиваться с несколькими видами ошибок.

Во-первых, это синтаксические ошибки, связанные с неправильным употреблением различных элементов и конструкций языка Паскаль. Причиной возникновения таких ошибок обычно являются недостаточно хорошее знание языка программирования и опечатки при наборе текста программы. Такие ошибки определяются уже на этапе компиляции и серьезной опасности не представляют.

Во-вторых, Паскаль является языком *программирования со строгим контролем типов*. Все объекты данных, используемые в программе, должны быть описаны в разделе описаний поэтому, если при наборе программы вследствие опечатки появляется «ложная переменная», компилятор обязательно сообщит об этом.

Третий вид ошибок доставляет программисту гораздо больше неприятностей. Это ошибки при выполнении программы. Сообщение о такой ошибке имеет вид:

Run-time error <errnum>at:<offset> - адрес в памяти, где произошла ошибка. Довольно часто оказывается, что синтаксически правильная программа завершает свое выполнение аварийно, с сообщением, например, о попытке деления на ноль.

Кроме того, программа может по непонятной причине зациклиться. Возникновение ошибок такого рода связанно с тем, что программа в процессе выполнения ведет себя не так, как предполагал программист. Почему? Выяснить это можно, применяя методы отладки программы.

Завершить работу «зациклившейся» программы можно нажатием клавиш Ctrl+Break.

Наиболее тяжелые ошибки бывают связаны с неправильным выбором модели, алгоритма решения задачи или с неправильной постановкой задачи. Здесь мы остановимся на тех методах отладки, которые доступны из интегрированной среды Турбо Паскаля. Речь

пойдет о возможностях встроенного отладчика, доступ к функциям которого открывает меню **Debug**.

Интегрированный отладчик Турбо Паскаля дает возможность пошагового выполнения программы. При этом можно просматривать значения различных переменных, что иногда дает ценную информацию о реальной работе программы.

Для запуска сеанса отладки необходимо выбрать команду **Run>Trace into** или нажать клавишу **F7**. При этом программа вначале компилируется, а затем начинается ее пошаговое выполнение. Каждый шаг заключается в выполнении очередной строки операторов, и происходит он при очередном нажатии на **F7**.

Метод пошагового выполнения программы неудобен, если исходный текст программы имеет большой объем, а программиста интересуют значения переменных только в избранных местах программы, да и то не всегда, а при выполнении определенных условий. В этом случае на помощь приходит команда меню **Debug> Breakpoints**, позволяющая разместить в тексте программы «точки прерывания».

Для каждой точки прерывания указывается номер строки, где она устанавливается, условие при выполнении, которого программа приостановится в указанном месте, а также количество прохождений точки до ее «срабатывания». Ставшие ненужными точки прерывания можно удалить, воспользовавшись той же командой.

Пример сообщений об ошибках, выдаваемых на экран

Error 3: Unknown identifier.	Неизвестный идентификатор
Error 4: Duplicate identifier	Двойной идентификатор
Error 5: Syntax error	Синтаксическая ошибка
Error 26: Type mismatch	Несоответствие типов, например, в операторе присваивания.
Error 36: Begin expected	Отсутствует начало составного оператора **begin**
Error 90 “=” expected	Отсутствует знак =