МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ЕЛЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. И.А. БУНИНА»

И.И. Васильева, О.Ю. Мелякова

СТРУКТУРНОЕ И ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Учебное пособие

Елец -2016

УДК….
ББК……
……..

Печатается по решению редакционно-издательского совета
Елецкого государственного университета им. И.А. Бунина
от _____, протокол №____

Рецензенты:
И.Н. Тарова, кандидат педагогических наук, доцент кафедры Прикладной математики и информатики, заместитель директора по учебно-производственной работе Центра СПО Елецкого государственного университета им. И.А. Бунина
Н.В. Позднякова, учитель информатики МБОУ СОШ №24

И.И. Васильева, О.Ю. Мелякова
_____ Структурное и объектно-ориентированное программирование: учебное пособие. – Елец: Елецкий государственный университет им. И.А. Бунина, 2016. - ____ с.

В учебном пособии описаны базовые понятия алгоритмизации и программирования, а также визуальные среды программирования. В пособии приводятся как общие сведения, характерные для большинства языков и сред программирования, так и специфичные приемы работы с современными упрощенными аналогами классических систем

Данное учебное пособие предназначено для студентов СПО, обучающихся по специальностям 09.02.03 – «Программирование в компьютерных системах», 09.02.02 – «Компьютерные сети». Оно может быть использовано учителями информатики в рамках факультативных занятий в старших классах, а также преподавателями в колледжах и училищах.

УДК…..
ББК……

Оглавление

Введение
Глава 1. Базовые понятия алгоритмизации и программирования
1.1 Переменные, константы, типы
1.2 Арифметические выражения и математические функции
1.3 Логические, строковые выражения
1.4 Линейные программы, ввод-вывод
1.5 Процедуры и функции пользователя
1.6 Ветвления, множественный выбор
1.7 Циклы
1.8 Массивы
1.9 Структуры данных
1.10 Файлы
1.11 Графика.
Глава 2. Визуальные среды программирования
2.1. Формы, проекты
2.2. Основные компоненты
Глава 3. Решение прикладных задач
3.1. Линейные программы
3.2. Разветвляющиеся программы
3.3. Циклические программы
3.4. Одномерные массивы
3.5. Сортировка массивов методом «пузырька»

Введение

Учебников по программированию написано великое множество, но практически все они имеют ряд недостатков. Долгое время в качестве основных средств структурного программирования использовались QBasic, Turbo Pascal и Borland C++, функционирующие в среде MS DOS. На смену им пришли визуальные инструменты, реализующие принципы объектно-ориентированного программирования. Однако, начинающим программистам сложно сразу переходить к ООП без знания базовых основ языка. Многие авторы предлагают начинать работу с консольного режима, имитирующего окно вывода результатов MS DOS. Но некоторые операторы, особенно связанные с вводом-выводом, потребуют дополнительных слов в синтаксическом описании. Немаловажным фактором является высокая цена известных программных систем (Delphi, Microsoft Visual Studio и т.д.), поэтому ряд составителей учебно-методических пособий опирается на среды программирования, работающие в ОС Linux.

В настоящее время появилась возможность использовать компактные бесплатные и условно-бесплатные инструментальные среды, в том числе функционирующие с помощью web-интерфейса. Поэтому в данном пособии приводятся как общие сведения, характерные для большинства языков и сред программирования, так и специфичные приемы работы с современными упрощенными аналогами классических систем.

Язык программирования – это язык для записи программ для вычислительных машин и устройств.

Классификация языков программирования №1

1) ЯП низкого уровня (машинные коды и ассемблеры)
2) ЯП высокого уровня
3) ЯП сверхвысокого уровня (логические и т.д.)

Классификация языков программирования №2

1) Императивные (коды)
2) Декларативные (логические)

Классификация языков программирования №3

1) Структурные
2) Объектно-ориентированные

Классификация языков программирования №4

1) интерпретаторы
2) компиляторы

Классификация систем (сред) программирования

1) Консольные
2) Визуальные

Глава 1. Базовые понятия алгоритмизации и программирования

1.1 Переменные, константы, типы

Переменная – это элемент программы, предназначенный для хранения данных в процессе выполнения программы. Переменная представляет собой зарезервированное место в оперативной памяти для временного хранения данных. Каждая переменная имеет имя и значение. Имя переменной уникально и не может меняться в процессе выполнения программы. Значение переменной может многократно меняться в процессе выполнения программы.

Это определение имеет место только для императивных языков программирования, а для декларативных переменная представляет собой средство для поиска и сравнивания информации по запросу с известными фактами.

Имя переменной – это строка символов, которая отличает эту переменную от других элементов программы. Иначе имя переменной называют идентификатор (от английского identify – распознавать, устанавливать идентичность). Имя переменной задается программистом. Оно должно подчиняться правилу имен и быть уникальным. Правило имен состоит из следующих пунктов:

1. В имени переменной можно использовать буквы, цифры и знак подчеркивания.
2. Первым символом имени должна быть буква.
3. Остальные символы имени – буквы, цифры и знак подчеркивания.
4. Имя переменной не должно содержать пробелы, скобки, знаки препинания и математических операций.
5. Длина имени не должна превышать 255 символов.
6. Имя переменной не должно совпадать ни с одним ключевым словом системы программирования.
7. Некоторые языки программирования чувствительны к регистру (например, C++), другие – нет (например, Бейсик), но лучше придерживаться правила единого написания строчных или прописных букв.
8. В ряде систем в именах переменных допускается использование букв кириллицы, но во избежание путаницы лучше использовать только буквы латинского алфавита.
9. Для небольших программ проще обозначать идентификаторы 1-2 символами, например, x или a1, а для сложных – с большим количеством подпрограмм и вычислительных процессов – лучше использовать полные имена конкретных данных, например, summa или FirstDay, выделяя прописными буквами или знаком подчеркивания начало слова.

При решении задач выполняется обработка информации различного свойства, например, дробные и целые числа, слова, строки и т.д. Для описания множества допустимых значений величины и совокупности операций, в которых участвует данная величина, используется указание ее типа данных.

Тип данных – это множество величин, объединенных определенной совокупностью допустимых операций. Каждый тип имеет свой диапазон значений и специальное зарезервированное слово для описания. Все типы данных можно разделить на две группы: скалярные (простые) и структурированные (составные). Простые типы данных также делятся на стандартные и пользовательские. Стандартные – предлагаются разработчиками языка программирования, а пользовательские разрабатывают сами программисты. Таким образом, типом данных называется способ хранения и представления данных в компьютерной системе. Он задает размер ячейки, в которую будет записано то или иное значение, определяя тем самым его максимальную величину или точность задания.

Количество типов и методы их описания в программе зависят от конкретной системы программирования. Так, например, в устаревших версиях языка Basic использовались пять типов с суффиксной записью, т.е. Basic распознает тип по последнему символу в имени переменных:

Символьные (A$)

Вещественные двойной точности (A#)

Вещественные обычной точности (A!, A) - используется по умолчанию

Длинные целые (A&)

Целые (A%)

Применение суффиксов не всегда удобно: они загромождают текст, да и ошибиться в них легко. Поэтому в BASIC был предусмотрен другой способ описания типа переменной.

Общий формат команды описания типа:

DEFINT X-X (целые числа, INTeger)

DEFLNG X-X (длинные целые числа, LoNG)

DEFSNG X-X (вещественные числа обычной точности, SiNGle)

DEFDBL X-X (вещественные числа двойной точности, DouBLe)

DEFSTR X-X (строки символов, STRing)

Комбинация X-X - диапазон букв. Вместо диапазона можно указать одну букву.

Например:

DEFLNG A-D, все переменные, имена которых начинаются с букв, лежащих в указанном диапазоне, т.е. с A до D, будут считаться длинными целыми (aRc, BT, DLINA и т.п.)

DEFSTR STR, переменная STR- строка символов.

В классическом Бейсике тип данных было указывать необязательно, а в языке Паскаль все переменные должны быть строго типизированы. В связи с увеличением типов данных и концепцией обязательного указания типа предыдущие способы объявления переменных неудобны, поэтому в настоящее время применяется словесное указание типа с одновременным резервированием памяти для значения переменной. Т.е. в наименованиях переменных можно использовать префиксы, отражающие тип переменной. При таком обозначении переменных повышается читабельность программы и снижается

количество ошибок программирования. Префиксы отражают тип переменной и область ее действия.

Например, в системе Турбо Паскаль к простым типам относятся порядковые и вещественные типы. Порядковые типы отличаются тем, что каждый из них имеет конечное число возможных значений. Эти значения можно определенным образом упорядочить (отсюда - название типов) и, следовательно, с каждым из них можно сопоставить некоторое целое число - порядковый номер значения.

Вещественные типы, строго говоря, тоже имеют конечное число значений, которое определяется форматом внутреннего представления вещественного числа. Однако количество возможных значений вещественных типов настолько велико, что сопоставить с каждым из них целое число (его номер) не представляется возможным.

К порядковым типам относятся целые, логический, символьный, перечисляемый и тип-диапазон. Вещественные (действительные) типы данных представляют собой значения, которые используются в арифметических выражениях и могут быть представлены двумя способами: с фиксированной и с плавающей точкой.

Действительные числа с фиксированной точкой записываются по обычным правилам арифметики, только целая часть от дробной отделяется точкой. Если точка отсутствует, число считается целым. Перед числом может стоять знак «+» или «-». Если знака нет, то число считается положительным.

Числа в форме с плавающей точкой представляются в экспоненциальном виде: mE+p, где m – мантисса (целое или дробное число), E означает 10 в степени, p – порядок (целое число).

Например, 5.18E+2 = 5.18 * 102 = 518

10E-03 = 10 * 10 –3 = 0.01

Литерный (символьный) тип char определяется множеством значений кодовой таблицы ПК. Каждому символу приписывается целое число в диапазоне от 0 до 255. Для кодировки используется код ASCII. Например, код символа 'A' при русской раскладке клавиатуры будет равен 192.

Для размещения в памяти переменной литерного типа нужен 1 байт.

Логический (булевский) тип boolean определяется двумя значениями: true (истина) и false (ложь). Он применяется в логических выражениях и выражениях отношения. Для размещения в памяти - 1 байт.

Перечисляемый тип задается перечислением тех значений, которые он может получать. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамленном круглыми скобками, например:

```
type
colors =(red, white, blue);
```

Применение перечисляемых типов делает программы нагляднее.

Соответствие между значениями перечисляемого типа и порядковыми номерами этих значений устанавливается порядком перечисления: первое значение в списке получает порядковый номер 0, второе - 1 и т.д. Макси-

мальная мощность перечисляемого типа составляет 65536 значений, поэтому фактически перечисляемый тип задает некоторое подмножество целого типа WORD и может рассматриваться как компактное объявление сразу группы целочисленных констант со значениями 0, 1 и т.д.

Использование перечисляемых типов повышает надежность программ благодаря возможности контроля тех значений, которые получают соответствующие переменные.

Типы языка C++ так же можно разделить на основные и составные.

К основным типам данных языка относят:

· char символьный ;

· int целый ;

· float с плавающей точкой ;

· double двойной точности ;

· bool логический .

Для формирования других типов данных используют основные типы и так называемые спецификаторы. Типы данных созданные на базе стандартных типов с использованием спецификаторов называют составными типами данных. В C++ определены четыре спецификатора типов данных:

· short короткий ;

· long длинный ;

signed знаковый ;

· unsigned беззнаковый .

В системе Турбо Паскаль рассматриваются структурированные типы, к которым относятся:

Строковый тип

Регулярный тип

Множественный тип

Файловый тип

В отличие от него, в языке C++ первые три типа отсутствуют, а работа с файлами имеет несколько другую природу.

Для связи со структурными языками в языке Бейсик были пересмотрены методы описания типов. Так, в Visual Basic выделено 16 типов данных, которые делятся на две основные группы: числовые и нечисловые. Числовые типы данных предназначены для хранения и обработки чисел. Они в свою очередь делятся на целые типы и рациональные. Нечисловые типы данных предназначены для хранения нечисловой информации. В эту категорию попадают символьные типы данных, логические и прочие.

Несмотря на то, что в декларативных языках переменная имеет иной смысл, в Прологе существует раздел для описания доменов (различных классов объектов, используемых в программе).

Все данные в Python представлены объектами. Имена являются лишь ссылками на эти объекты и не несут нагрузки по декларации типа. Значения встроенных типов имеют специальную поддержку в синтаксисе языка: мож-

но записать литерал строки, числа, списка, кортежа, словаря (и их разновидностей). Синтаксическую поддержку операций над встроенными типами можно легко сделать доступной и для объектов определяемых пользователей классов.

Следует также отметить, что объекты могут быть неизменчивыми и изменчивыми. Например, строки в Python являются неизменчивыми, поэтому операции над строками создают новые строки.

Рассмотрим характеристики каждого типа данных в общей таблице.

Язык или система программирования	Назначение типа	Тип данных	Диапазон значений	Занимаемая память (байт)
Паскаль (Turbo Pascal, PascalABC, Free Pascal, Delphi, Lazarus). В первоначальном языке Паскаль было 3 основных типа (Integer, Real, Char).	целый	Byte	0..255	1
		ShortInt	-128..127	1
		Integer	-32768..32767	2
		Word	0..65535	2
		longInt	-2147483648 ..2147483647	4
	вещественный	Real	2.9*10E-39 .. 1.7*10E38	6
		Single	1.5*10E-45 .. 3.4*10E38	4
		Double	5.0*10E-324 .. 1.7*10E308	8
		Extended	1.9*10E-4951 .. 1.1*10E4932	10
		Comp	-2E+63+1 .. 2E+63-1	8
	логический	Boolean	True, False	1
	символьный (литерный)	Char	0..255	1
С С++ Логический тип отсутствовал в первоначальном языке Си.	целый	Int или Signed Int	-32768..32767	2-4 в зависимости от разрядности процессора
		Unsigned Int	0..65535	4
		Short Int, Signed Short Int	-32768..32767	2
		Long	-2147483647 .. 2147483647	4

		Int или Signed Long Int		
		Un-signed Short Int	0 .. 65535	2
		Long Long Int	$-(2^{63}-1)$ .. $(2^{63}-1)$	8
		Un-signed Long Int	0 .. 4294967295	4
		Un-signed Long Long Int	0 .. $2^{64}-1$	8
	веще-ственный	Float	3.4E-38 .. 3.4E+38	4
		Double	1.7E-308 .. 1.7E+308	8
		Long Double	3.4E-4932 .. 3.4E+4932	10
	логиче-ский	Bool	0 или 1	1
	символь-ный	Char	-128..127	1
		Un-signed Char	0..255	1
Бейсик (QBasic, Small Basic, Visual Basic, VBA)	целый	SByte	-128..127	1
		Short	-32 768 .. 32 767	2
		Integer	-2 147 483 648 .. 2 147 483 647	4
		Long	-9 233 372 036 854 775 808 .. 9 233 372 036 854 775 807	8
		Byte	0..255	1
		UShort	0..65535	2
		UInte-ger	0 .. 4 294 967 295.	4
		ULong	0 .. 18 446 744 073 709 551 615.	8
	веще-ственный	Single	-3.4028235E38 .. -1.401298E-45; 1.401298E-45 ..	4

			3.4028235E38	
		Double	-1.797693134862E308 .. -4.9406564584146E-324; 4.94065645841246544E-324 .. 1.79769313486231570E308	8
		Deci-mal	-7,9228162514264337593543950335 .. 7,9228162514264337593543950335	16
	логиче-ский	Boole-an	True (1), False (0)	Зависит от систе-мы
	симво-льный	Char	Символ в кодировке Unicode	2
		String	0 .. 2000000000 символов	различно
	Дата и время	Date	Дата от 1.01.0001 г до 31.12.9999 г и время от 0:00:00 до 23:59:59	8
	Объект-ный	Object	Ссылка на объект любого типа	Различна
Python	целый	Int	-2147483648 .. 2147483647	
		Long		Зависит от объема памяти
		Bool	True, False	
	веще-ственный	Float		
	ком-плексный	Com-plex		
	строковы й	Str		
		Unicod e		
		Tuple		
Пролог (Turbo Prolog, Visual Prolog)	целый	Integer	–32768 .. 32767	4
	веще-ственный	Real	+1E-307 .. +1E308	8
	симво-льный	Char		Зависит от систе-мы
	строко-вый	String	0..250 символов	
	Симво-лическое	Symbol	Последовательность симво-лов	

	имя			
	файло-вый	File	Имя файла	

Объявление переменных

1) Паскаль (Delphi, Lazarus): var name of type; (специальный раздел перед основной программой);

2) C (C++, C#): type name; (в тексте основной программы до использования);

3) Basic (Visual Basic, VBA): dim name as type; (в старых версиях – необязательно, в новых – по следующей схеме):

Dim ИмяПеременой [As тип]

Static ИмяПеременой [As тип]

Private ИмяПеременой [As тип]

Public ИмяПеременой [As тип]

Ключевые слова Dim, Static, Private, Public влияют на область видимости переменной.

VBA не требует явного описания переменных перед их использованием. Переменные, которые сразу используются в программе без предварительного описания, называются неявно описанными. Неявно описанным переменным присваивается тип данных Variant. Однако, рекомендуется явно описывать все переменные, так как это способствует увеличению скорости выполнения программы и упрощает ее отладку. Для описания переменных в VBA используются операторы Dim, Public, Private и Static.

Параметр имя_переменной – имя описываемой переменной. Необязательные скобки и параметр индексы используются для описания массивов. Дополнительный параметр As тип позволяет назначить переменной тип данных. Если необходимо явно задать тип, то это надо сделать для каждой переменной списка.

Область видимости и время жизни переменной определяется тем, где и как была описана переменная:

– Личная переменная уровня процедуры описывается с помощью инструкции Dim, размещенной в процедуре. Такая переменная сохраняет свое значение только при выполнении процедуры, а при повторном запуске процедуры ее необходимо инициализировать заново.

– Переменная уровня процедуры, описанная с помощью инструкции Static, также доступна только в той процедуре, где она описана. Однако при выходе из процедуры ее значение сохраняется.

– Личная переменная уровня модуля описывается с помощью инструкции Dim или Private, размещенной в разделе описаний модуля.

– Общая переменная, доступная во всех модулях проекта, описывается с помощью инструкции Public в разделе описаний модуля.

Для того чтобы VBA требовал явного описания каждой используемой переменной, в раздел описаний модуля необходимо включить инструкцию Option Explicit.

4) Prolog: domains name=type;

5) В Python нет необходимости указывать тип, т.к. понятия переменной как таковой здесь нет. В Питоне есть ссылки на объекты, которые автоматически «знают» свой тип. Однако, если задать тип перед выражением или идентификатором явно, то будет использоваться функция преобразования типа.

1.2 Арифметические выражения и математические функции

Оператор присваивания используется для присваивания элементу данных языка (переменной, константе, элементу массива, свойству объекта) значения.

Оператору присваивания в VB/VBA соответствует знак =(равно).

Синтаксис: ИмяПеременной = Выражение

После выполнения оператора присваивания переменной с именем ИмяПеременной будет присвоено значение, полученное в результате вычисления выражения, стоящего справа от знака =(равно).

Тип данных переменной ИмяПеременной должен быть совместим с типом данных вычисленного выражения.

Допускается запись нескольких операторов в строке. В качестве разделителя операторов используется знак :(двоеточие).

Оператор присваивания в Паскаль имеет вид ИмяПеременной := Выражение.

Типы данных так же, как и в предыдущем случае, должны быть совместимы. В качестве разделителя операторов используется знак ; (точка с запятой).

Синтаксис оператора присваивания в языке Си/C++ такой же, как для Бейсика, однако, типы данных уравниваются автоматически. Например, если операнды вещественного типа, а результат заявлен как число целого типа, то полученное значение округляется до целого, т.е. ошибки не происходит. В C++ существует возможность присваивания нескольким переменным одного и того же значения. Такая операция называется множественным присваиванием и в общем виде может быть записана так:

имя_переменной1= имя_переменной2=..= имя_переменнойN=значение;

Операции +=, - =, *=, /= называют составным присваиванием. В таких операциях при вычислении выражения, стоящего справа, используется значение переменной из левой части, например:

x+=p; //Увеличение x на p, то же что и x=x+p.

x-=p; //Уменьшения x на p, то же что и x=x-p.

x*=p; //Умножение x на p, то же что и x=x*p.

x/=p; //Деление x на p, то же что и x=x/p.

Такой вид может использоваться в системах PascalABC.Net, Visual Basic

В Питоне используется часто множественное присваивание.

Основные операции

Операция	Система программирования			
	Паскаль	Си	VBA VB	Питон
Сложение	+	+	+	+
Вычитание	-	-	-	-
Умножение	*	*	*	*
Деление	/	/	/	/
Возведение в степень	Sqr(x) (только x^2)	Pow(x,y)	^	Pow(x,y) **
Целочисленное деление	div	нет	\	//
Остаток от деления	mod	%	mod	%
Инкремент	Inc(x)	x++ ++x		
Декремент	Dec(x)	x-- --x		
Больше	>	>	>	>
Меньше	<	<	<	<
Равно	=	==	=	==
Не равно	<>	!=	<>	!=

Основные математические функции

Функция	Система программирования			
	Паскаль	Си	VBA	VB
Случайное число от 0 до x	Random(x)		Rnd(x)	
Целая часть числа	Int(x)		Int(x)	Math. Truncate(x)
Модуль числа	Abs(x)	Abs(x) Fabs(x)	Abs(x)	Math.Abs(x)
Квадратный корень	Sqrt(x)		Sqr(x)	Math.Sqrt(x)
Синус числа	Sin(x)	Sin(x)	Sin(x)	Math.Sin(x)
Косинус числа	Cos(x)	Cos(x)	Cos(x)	Math.Cos(x)
Тангенс числа		Tan(x)	Tan(x)	Math.Tan(x)
Арктангенс числа	Arctan(x)	Atan(x)	Atn(x)	Math.ATan(x)
Экспонента (e^x)	Exp(x)	Exp(x)	Exp(x)	Math.Exp(x)
Натуральный логарифм	Ln(x)	Log(x)	Log(x)	Math.Log(x)
Десятичный логарифм		Log10(x)		Math.Log10(x)
Определение знака числа			Sgn(x)	Math.Sign(x)
Округление числа	Round(x)		Round(x,a)	

1.3 Логические, строковые выражения

Логическая операция	Система программирования		
	VB/VBA	Паскаль	C++
Отрицание	NOT	NOT	!
Логическое И	AND	AND	&&
Логическое ИЛИ	OR	OR	||
Исключающее ИЛИ	XOR	XOR	^
Эквивалентность	EQV		
Импликация	IMP		

В Питоне набор математических и логических функций зависит от используемого модуля. В Прологе часть функций описывается предикатами.

Символьные и строковые величины

В некоторых языках есть оба типа, в других строка определяется как массив символов.

Литерный (символьный) тип char определяется множеством значений кодовой таблицы ПК. Каждому символу задается целое число от 0 до 255. В программе значения переменных и констант типа char должны быть заключены в апострофы.

Над данными символьного типа определены операции отношения: =, <>, >,<,<=,>=, вырабатывающие результат логического типа, и следующие стандартные функции:

Chr(x) – преобразует выражение x в символ и возвращает значение символа

Ord(ch) – преобразует символ ch в его код и возвращает значение кода

Pred(ch) – возвращает предыдущий символ

Succ(ch) – возвращает следующий символ

Строка (строковый тип данных) в языке Паскаль – это последовательность символов кодовой таблицы ПК. Количество символов в строке (длина строки) может лежать в диапазоне от 0 до 255. Для определения данных строкового типа используется идентификатор string, за которым следует значение максимальной длины строки данного типа (заключается в квадратные скобки).

Строковые данные могут использоваться в качестве констант. Строковая константа – последовательность символов, заключенная в апострофы.

Переменную строкового типа можно определить в разделе описания переменных:

Var <имя>: string[<максимальная длина строки>].

В описании строки можно не указывать длину, в этом случае она равна максимальной величине – 255. Элементы строки определяются именем стро-

ки с индексом, заключенным в квадратные скобки. Например, N[5]. Первый символ строки имеет номер 1 и т.д. Можно сказать, что строка представляет собой одномерный массив, элементами которого являются символы. Тип string и тип char совместимы, они могут употребляться в одних и тех же выражениях.

Выражения, в которых операндами служат строковые данные, называются строковыми. Они могут состоять из строковых констант, переменных, знаков операций. Над этими данными допустимы операция сцепления (конкатенация) и операции отношения.

Операция сцепления (+) применяется для соединения нескольких строк в одну строку. Сцеплять можно и константы, и переменные. Длина результирующей строки не должна превышать 255 символов.

Операции отношения (=, <>, >, <, <=, >=) проводят сравнение двух строк и имеют приоритет более низкий, чем операция конкатенации. Сравнение строк производится слева направо до первого несовпадающего символа. Строка считается больше, если в ней первый несовпадающий символ имеет больший номер в таблице кодов.

Если строки имеют различную длину, но в общей части символы совпадают, то более короткая строка меньше. Строки равны, если они полностью совпадают.

Для обработки строковых данных можно использовать специальные процедуры и функции.

Процедура Delete(St, poz, n) – удаление n символов строки St, начиная с позиции Poz.

Процедура Insert (S1, S2, Poz) – вставка строки S1 в строку S2, начиная с позиции Poz.

Процедура Str(N,St) – преобразование числового значения N в строковый и помещение результата в строку St.

Процедура Val(St, N,Code) – преобразует значение St в величину целочисленного или вещественного типа и помещает результат в N. Code – целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение Code равно 0, если же обнаружена ошибка, то Code будет содержать номер позиции первого ошибочного символа, а значение N не определено.

Функция Copy(S, Poz, N) – выделяет из строки S подстроку длиной N символов, начиная с позиции Poz.

Функция Concat(S1,S2,…,Sn) – выполняет сцепление строк S1,S2,…,Sn в одну строку.

Функция Length(S) – определяет текущую длину строки S.

Функция Pos(S1,S2) – определяет первое появление в строке S2 подстроки S1.

Функция UpCase (ch) – преобразует строчную букву в прописную. Обрабатывает буквы только латинского алфавита.

Строка в C++ – это последовательность символов. Если в выражении встречается одиночный символ, он должен быть заключен в одинарные кавычки. При использовании в выражениях строка заключается в двойные кавычки. Признаком конца строки является нулевой символ ‘\0’. В C\C++ в отличии от других языков программирования отсутствует тип данных строка, строки в Си можно описать с помощью массива символов (массив элементов типа char), в массиве следует предусмотреть место для хранения признака конца строки ('\0').

Основные функции обработки строк расположены в библиотеках:

1) библиотека обработки символов ctype.h;

2) библиотека преобразования строк stdlib.h;

3) библиотека обработки строк string.h.

Например, функция strlen(const char *s) вычисляет длину строки s в байтах.

Часть функций работы со строками находится в библиотеке ввода/вывода stdio.h.

Строка в Visual Basic– это некоторый набор символов, в том числе и пустой. Для обозначения строки используются кавычки ("). Пустая строка обозначается парой кавычек, между которыми нет ни одного символа, в том числе и пробела (s ="").

Строки описываются с помощью типа String. Каждая строка может содержать до двух миллиардов символов в формате Unicode. Объем памяти, занимаемой строковой переменной, зависит от количества символов в этой строке. Чем больше символов, тем больше памяти требуется для хранения этой строки.

Число символов в строке называется длиной строки. Длина пустой строки равна нулю. Все символы в строке последовательно пронумерованы. Нумерация идет слева направо и начинается с единицы. Номер символа в строке также называют позицией символа.

Подстрока – это любой фрагмент строки, состоящий хотя бы из одного символа исходной строки. Левой подстрокой называется такая подстрока, которая начинается с первого символа исходной строки. Правая подстрока – это подстрока, которая заканчивается последним символом исходной строки.

Конкатенацией двух строк s1 и s2 называется строка s, для которой s1 является левой подстрокой, s2 – правой подстрокой, а длина строки s равна сумме длин строк s1 и s2. Часто конкатенацию называют сложением или склейкой строк. В Visual Basic она обозначается знаком плюс.

s = s1 + s2

Строковое выражение – это либо отдельная строка, либо строка и строковое выражение, между которыми стоит знак конкатенации.

Основные функции обработки строк:

- Strings.Len(Строка) – возвращает количество символов в строке, то есть ее длину.

- Strings.Left(Строка, Длина) – выделяет левую подстроку указанной Длины из заданной Строки.
- Strings.Right(Строка, Длина) – выделяет правую подстроку указанной Длины из заданной Строки.
- Strings.Mid(Строка, Позиция, Длина) – выделяет подстроку заданной Длины из исходной Строки, начиная с указанной Позиции. Параметр Длина может отсутствовать. В этом случае функция Strings.Mid возвращает правую подстроку, которая начинается с заданной Позиции. Функция Mid может стоять не только справа от знака присваивания, но и слева. В этом случае префикс Strings не ставится. Если функция Mid стоит слева от знака присваивания, то соответствующая подстрока исходной Строки будет заменена на значение строкового выражения, стоящего в правой части оператора присваивания. При этом длина исходной строки не меняется.

Существует несколько способов обрабатывать строку символов в Visual Basic. Один из них – посимвольная обработка. В этом случае строка рассматривается как массив символов. Соответственно, для ее обработки организуется цикл по всем символам строки. Так как нумерация символов начинается с единицы, то начальное значение счетчика берут равным единице, а не нулю, как при работе с одномерным массивом. Номер последнего символа совпадает с длиной строки, поэтому конечное значение счетчика записывают в виде Len(s), где s – обрабатываемая строка. В теле цикла строку обрабатывают так же, как и одномерный массив. С помощью функции Strings.Mid выделяют один символ, его анализируют и обрабатывают.

Строка – это набор символов. При программировании на Прологе символы могут быть «записаны» при помощи алфавитно-цифрового представления или при помощи их ASCII-кодов. Обратный слэш (\), за которым непосредственно следует ASCII-код (N) символа, интерпретируется как символ. Для представления одиночного символа выражение \N должно быть заключено в апострофы (‘\N’). Для представления строки символов ASCII-коды помещаются друг за другом и вся строка заключается в кавычки («\N\N\N»).

Операции, обычно выполняемые над строками, включают:

- объединение строк для образования новой строки;
- разделение строки для создания двух новых строк, каждая из которых содержит некоторые из исходных символов;
- поиск символа или подстроки внутри данной строки.

Для удобства работы со строками Турбо-Пролог имеет несколько встроенных предикатов, манипулирующих со строками:

- str_len – предикат для нахождения длины строки;
- concat – предикат для объединения двух строк;
- frontstr – предикат для разделения строки на две подстроки;
- frontchar – предикат для разделения строки на первый символ и остаток;
- fronttoken – предикат для разделения строки на лексему и остаток.

В Python строки и символы нужно заключать в кавычки (одиночные или двойные). Элементы (символы) в строке нумеруются, начиная с нуля. Одиночный символ - буква - является с точки зрения Python строкой, состоящей из одного элемента.

Максимально возможное количество символов в строке (длина строки) в Python ограничивается только доступным объёмом памяти. Так что текст любого разумного размера (например, несколько тысяч страниц) может быть записан в одну строку Python.

Числа могут быть преобразованы в строки с помощью функции str ().

Если строка является последовательностью знаков-цифр, то она может быть преобразована в целое число в помощью функции int (), а в вещественное - с помощью функции float (). Для любого символа можно узнать его номер (код символа) с помощью функции ord(). И наоборот, получить символ по числовому коду можно с помощью функции chr().

1.4 Линейные программы, ввод-вывод

Структура программы:

1) Служебные операторы для добавления библиотек (модулей), описания программы или настроек компилятора (интерпретатора)

2) Раздел описания переменных и структур

3) Подпрограммы (процедуры, функции)

4) Основная программа (набор исполняемых команд)

5) Завершение программы, выгрузка модулей

Часто основная программа обрамляется операторными скобками или ключевыми словами.

Линейные программы

Команды выполняются друг за другом. Если они представляют собой математические выражения и формулы, то линейные программы схожи с режимом калькулятора. Не путать с понятием «линейное программирование», которое является методом оптимизации моделей и применяется для нахождения граничного значения целевой функции с заданными условиями в виде системы линейных неравенств.

Самая простая задача – найти сумму двух натуральных чисел, введенных с клавиатуры.

Если в выражениях присутствуют операция деления, извлечение квадратного корня или нахождение логарифма, то линейный алгоритм может привести к ошибке при неверно введенных данных. Поэтому на такие случаи, а также на ряд геометрических задач, лучше наложить условия и использовать разветвляющиеся алгоритмы.

Команды (процедуры или функции) ввода-вывода

Ввод и вывод означает, что программа должна сообщать результат своей работы потребителю: пользователю-человеку или другой программе (например, программе управления принтером).

В программировании существует специальное понятие консоль, которое обозначает клавиатуру при вводе и монитор при выводе.

Для того чтобы получить данные, вводимые пользователем вручную (т.е. с консоли), в Паскале применяются команды:

read (<список_ввода>)

readln (<список_ввода>)

Первая из этих команд считывает все предложенные ей данные, оставляя курсор в конце последней строки ввода, а вторая - сразу после окончания ввода переводит курсор на начало следующей строки. В остальном же их действия полностью совпадают.

Для того чтобы вывести на экран какое-либо сообщение, в Паскале воспользуйтесь процедурами:

write(<список_вывода>)

writeln(<список_вывода>)

Первая из них, напечатав на экране все, о чем ее просили, оставит курсор в конце выведенной строки, а вторая переведет его в начало следующей строчки.

Список вывода может состоять из нескольких переменных или констант, записанных через запятую; все эти переменные должны иметь тип либо базовый, либо строчный.

Переменные, составляющие список вывода, могут относиться к целому, вещественному, символьному или булевскому типам. В качестве элемента списка вывода кроме имен переменных могут использоваться выражения и строки.

Оператор Writeln без параметров реализует пропуск строки и переход к началу следующей строки.

Если для вывода информации просто перечислять переменные через запятую, то выводимые символы окажутся "слепленными". Чтобы этого не случилось, нужно позаботиться о пробелах между выводимыми переменными:

```
writeln(a, ' ', b, ' ', c);
```

Но предпочтительнее задать для всех (или хотя бы для некоторых) переменных формат вывода:

```
writeln(a:5,  b, c:20:5);
```

В классических вариациях Бейсика (QBasic, SmallBasic) используются встроенные команды ввода-вывода:

INPUT "Сообщение", список переменных

При выполнении команды INPUT вычисления приостанавливаются и на экран дисплея выводится поясняющее сообщение, если вы его написали. В списке переменных через запятую указываются имена переменных, которые принимают вводимые данные.

Курсор устанавливается следом за последним символом выведенного текста, и программа ожидает ввода данных. Вы должны через запятую набрать все данные и нажать клавишу {Enter}. Если строка символов замкну-

та не запятой, а точкой с запятой, INPUT выводит следом за текстом символ "?" и устанавливает курсор через пробел после "?".

Оператор вывода данных:

PRINT список_выражений

В поле операндов через запятую или через точку с запятой перечисляются выражения, значения которых надо вывести. Значения данных выводятся с текущей позиции курсора.

1. Плотный вывод - разделитель ";"

Перед значением числа выводится либо пробел, либо знак "минус".

2. Зональный вывод - разделитель ","

Если указана запятая, QBASIC выводит данные по зонам, каждая зона - 14 позиций.

Если программа постоянно работает с некоторым набором числовых или символьных констант, то можно объявить такой набор блоком данных:

DATA список констант

В списке констант через запятую указываются значения констант,

В программе можно записать произвольное число операторов DATA. В блок данных по порядку включаются все константы и в памяти создаётся специальный указатель блока данных. Во время работы программы этот указатель содержит порядковый номер константы в блоке данных. При запуске программы указатель показывает на первую константу из блока данных.

Для присвоения значений констант из блока данных переменным используется оператор READ:

READ список переменных

В списке переменных через запятую указываются имена переменных, которым присваиваются значения констант из блока данных. Типы переменных в списке READ должны соответствовать типам констант из блока данных.

VB/VBA содержит множество встроенных функций. К наиболее часто используемым относятся функции InputBox() и MsgBox(), предназначенные, соответственно, для ввода данных пользователя и для вывода сообщений.

Функция InputBox выводит на экран диалоговое окно, содержащее сообщение и поле ввода, устанавливает режим ожидания ввода текста пользователем или нажатия кнопки, а затем возвращает значение типа String, содержащее текст, введенный в поле.

Синтаксис функции:

InputBox(prompt[, title] [, default] [, xpos] [, ypos] _[, helpfile, context])

Единственный обязательный аргумент функции prompt задает строковое выражение, отображаемое как сообщение в диалоговом окне. Следующие аргументы указывают соответственно заголовок окна, значение, отображаемое в поле ввода по умолчанию, координаты вывода левого верхнего угла окна по горизонтали и вертикали, имя файла справки и контекстный номер раздела справки.

Пример использования функции InputBox:

s = InputBox(“Укажите значение параметра X”, “Ввод параметров”)

Данная инструкция выведет на экран диалоговое окно. При щелчке пользователя по кнопке OK в переменной s будет сохранено текстовое значение, введенное в поле ввода. Если пользователь щелкнет по кнопке Отмена, то функция вернет пустую строку.

Функция MsgBox выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа Integer, указывающее, какая кнопка была нажата.

Синтаксис функции:

MsgBox(prompt [, buttons] [, title] [, helpfile, context])

Одноименные аргументы имеют тот же смысл, что и для функции InputBox(). Аргумент buttons содержит числовое выражение, которые указывают число и тип отображаемых кнопок, тип используемого значка, основную кнопку и модальность окна сообщения. Обычно этот аргумент задается как сумма встроенных констант. Его значение по умолчанию равняется 0 и соответствует выводу модального на уровне приложения окна сообщения без значка с одной кнопкой OK.

Пример использования функции MsgBox:

n = MsgBox("Недопустимые параметры. Продолжить?", _

vbYesNo+vbCritical, "Ввод параметров")

Данная инструкция выведет на экран модальное на уровне приложения диалоговое окно со значком критического сообщения и двумя кнопками: Да, Нет.

В языке Си/C++ существует 2 библиотеки ввода-вывода. В стандартной библиотеке <stdio.h> функция:

printf(строка форматов, список выводимых переменных)

выполняет форматированный вывод переменных, указанных в списке, в соответствии со строкой форматов.

Функция:

scanf(строка форматов, список адресов вводимых переменных)

выполняет ввод переменных, адреса которых указаны в списке, в соответствии со строкой форматов.

Строка форматов содержит символы, которые будут выводиться на экран или запрашиваться с клавиатуры и так называемые спецификации. Спецификации - это строки, которые начинаются символом % и выполняют управление форматированием:

% флаг ширина . точность модификатор тип

Параметры флаг, ширина, точность и модификатор в спецификациях могут отсутствовать.

В потоковой библиотеке <iostream.h> в программе автоматически создаются объекты-потоки cin для ввода с клавиатуры и cout для вывода на экран, а так же операции помещения в поток << и чтения из потока >>. Например, cin>>x – ввести значение переменной. Cout<<x - вывести на экран

значение любой переменной или текст. Текст необходимо заключать в двойные кавычки, кроме того, допустимо применение специальных символов \t и \n. В качестве перехода на следующую строку после вывода можно использовать endl.

Ввод и вывод в Прологе организуется с помощью специальных предикатов чтения и записи, которые могут рассматриваться как аналоги соответствующих подпрограмм в языках Паскаль и Си.

1) readchar(Var) - считывает символ с входного потока (по умолчанию с клавиатуры) и присваивает его переменной Var;

2) readint(Var) - считывает целое число и присваивает его переменной Var;

3) readln(Var) - считывает символы с входного потока до нажатия клавиши Enter. Введенные символы присваиваются переменной Var, которая должна быть строковой (string) или символьной (symbol).

4) readreal(Var) - считывает вещественное число;

5) write(Arg1, Arg2, ...) - выводит значения аргументов на текущее устройство (по умолчанию, на экран дисплея). Аргументы Arg1, Arg2, ... могут быть константами или переменными, которым заранее присвоены требуемые значения.

6) nl - вызывает перевод каретки в начало следующей строки.

В предикате write можно использовать символы, начинающиеся со знака \. Они имеют специальные значения:

\k - символы, имеющие ASCII код числа k;

\n - возврат каретки и перевод строки;

\t - табуляция.

В языке Питон для ввода значений переменных будем использовать функции input () и raw_input(). В качестве аргумента этих функции рекомендуется использовать строку-подсказку (приглашение для ввода), в которой кратко описывается, какие данные и как необходимо сообщить программе.

Синтаксис: name=input()

При использовании функции input () числовые значения пишутся как обычно, а строковые нужно писать в кавычках (двойных или одиночных).

Для ввода только строковых значений в Python используется функция raw_input(). Её особенности во многом совпадают с функцией input (). Есть одна деталь - строковые значения при их вводе не нужно заключать в кавычки. Если с помощью raw_input() вводить числа, они преобразуются в строки.

Для вывода результатов работы в Питоне используется инструкция print, которая не является функцией. Использование инструкции (команды) print позволяет производить вычисления «на лету» и выводить одновременно (одним оператором) строки и числа.

1.5 Процедуры и функции пользователя

Любую сложную задачу можно представить как совокупность более простых подзадач, связанных между собой. Как правило, решение каждой подзадачи оформляется в виде самостоятельного законченного фрагмента

программы. Такой фрагмент, оформленный определенным образом, называется подпрограммой.

Подпрограммы делятся на два класса: процедуры и функции.

Процедура – самостоятельная часть программного кода, имеющая имя и параметры, выполняющая некоторую последовательность действий и изменяющая значения некоторых своих параметров.

Функция – это самостоятельная часть кода, имеющая имя и вычисляющая на основе своих параметров (аргументов) некоторое значение, которое затем передается вызывающей программе.

В системах, основанных на языках Паскаль и Бейсик, присутствуют оба вида подпрограмм. В языке Си – только функции. Для небольших программ структурного программирования (например, Turbo Pascal, Quick Basic) процедуры и функции можно не применять, а в объектно-ориентированных системах каждый объект описывается кодом в собственной процедуре.

Для использования подпрограммы-процедуры в Паскале необходимо сначала описать процедуру, а затем обращаться к ней (обращение к процедуре – отдельный оператор). Описание процедуры включает заголовок (имя) и тело процедуры. Заголовок состоит из зарезервированного слова procedure, имени процедуры и, заключенного в скобки, списка формальных параметров с указанием типа. Название «формальные» эти параметры получили в связи с тем, что в этом списке заданы только имена для обозначения исходных данных и результатов работы процедуры, а при вызове подпрограммы на их место будут поставлены конкретные значения. Тело процедуры – блок, по структуре аналогичный программе.

При создании программ, использующих процедуры, следует учитывать, что все объекты, которые описываются после заголовка в теле процедуры, называются локальными объектами и доступны только в пределах этой процедуры.

Все объекты, описанные в вызывающей программе, называются глобальными и являются доступными внутри процедур, вызываемых этой программой.

Общий вид описания процедуры:

```
Procedure <имя> (список формальных параметров, блок описания);
Const …;
   …          блок описания
Var ….;
begin
<операторы>
end;
```

Подпрограмма-функция в Паскале обрабатывает данные, переданные ей из главной программы, и затем возвращает полученный результат (в отличие от процедуры). Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово Function, имя, список формальных параметров (заключенный в скобки) и тип

возвращаемого функцией значения. Тело функции представляет собой локальный блок, по структуре сходный с программой. Общий вид описания функции:

Function <имя> (<параметры>): <тип результата>;
Const …;
… блок описания
Var ….;
begin
<операторы>
end;

В разделе операторов должен находиться, хотя бы один оператор, присваивающий имени функции значение. Обращение к функции осуществляется по имени с указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам и иметь тот же тип.

Подпрограмма - именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке C/C++ подпрограммы реализованы в виде функций. Функция принимает параметры и возвращает единственное скалярное значение. Синтаксис:

Заголовок_функции
{
тело_функции
}
Заголовок функции имеет вид
type имя_функции ([список параметров])
type - тип возвращаемого функцией значения;
список параметров - список передаваемых в функцию величин, которые отделяются запятыми, каждому параметру должен предшествовать его тип;

В случае, если вызываемые функции идут до функции main, структура программы будет такой:

директивы компилятора
...
Тип_результата f1(Список_переменных)
{
Операторы
}
...
Тип_результата fn(Список_переменных)
{
Операторы
} int main(_)
Список переменных
{

```
Операторы основной функции, среди которых могут операторы
вызова функций f1, f2, ..., fn
}
```

В случае, если вызываемые функции идут после функции main, структура программы будет такой (заголовки функций должны быть описаны до функции main()). Опережающие заголовки функций называют прототипами функций.

```
директивы компилятора
...
Тип_результата f1(Список_переменных);
Тип_результата f2(Список_переменных);
...
Тип_результата fn(Список_переменных);
int main(Список_переменных)
{
Операторы основной функции, среди которых могут операторы
вызова функций f1, f2, ..., fn
}
Тип_результата f1(Список_переменных)
{
Операторы
}
...
Тип_результата fn(Список_переменных)
{
Операторы
}
```

Для того, чтобы функция вернула какое-либо значение, в ней должен быть оператор return значение;

Для вызова функции необходимо указать имя функции и в круглых скобках список передаваемых в функцию значений. Параметры, указанные в заголовке функции, называются формальными. Параметры, передаваемые в функцию, называются фактическими.

При обращении к функции фактические параметры передают свое значение формальным и больше не изменяются. Типы, количество и порядок следования формальных и фактических параметров должны совпадать. С помощью оператора return из функции возвращается единственное значение.

Для того чтобы функция возвращала не только скалярное значение, то в качестве передаваемого в функцию значения можно использовать указатель.

В свою очередь, процедуры в среде Visual Basic делятся на общие процедуры и процедуры обработки событий. Процедура обработки события вызывается операционной системой после наступления конкретного события. Общая процедура начинает работать только после ее явного вызова. После

выполнения общей процедуры происходит автоматический возврат управления в то место программы, откуда она была вызвана.

Параметры подпрограммы делятся на входные и выходные.

Входной параметр – это переменная, значение которой должно быть известно до начала работы программы. Она используется в процессе работы программы.

Выходной параметр – это переменная, значение которой вычисляется или изменяется в процессе работы подпрограммы. Заметим, что один и тот же параметр может одновременно быть и входным и выходным.

Описание процедуры:

```
Область видимости Sub Имя процедуры _
                (Список параметров)
      Операторы
   End Sub
```

Область видимости задается одним из двух ключевых слов: Private или Public. Она определяет доступность процедуры из других модулей проекта. Процедура, описанная как Private, доступна только в том модуле, в котором она описана. Процедура, описанная как Public, доступна во всех модулях проекта.

Sub – ключевое слово Visual Basic 2005, показывающее, что данная подпрограмма является процедурой.

Имя процедуры – уникальное название процедуры, построенное по правилу имен, которое отличает данную подпрограмму от всех других элементов проекта.

Список параметров – это перечисленные через запятую входные и выходные параметры. Количество параметров может быть произвольным. Комбинация входных и выходных параметров может быть любой. У процедуры могут быть только входные параметры, могут быть только выходные параметры. Список параметров может вообще отсутствовать, но круглые скобки ставить все равно необходимо. Каждый параметр в списке описывается отдельно по следующей схеме.

Режим передачи Имя параметра As Тип данных

Режим передачи параметра определяет способ передачи данных в подпрограмму. Существует два способа передачи: по ссылке и по значению.

Имя параметра – уникальное имя переменной, отличающее этот параметр от других параметров данной подпрограммы. Тип данных показывает, к какому типу данных принадлежит данный параметр.

Начиная со следующей после заголовка процедуры строки, пишутся операторы подпрограммы, которые реализуют необходимую последовательность действий. Среди этих операторов могут встречаться два особых оператора: Return и Exit Sub. Оба оператора немедленно прекращают выполнение данной подпрограммы и возвращают управление в вызывающую программу.

Описание процедуры заканчивается ключевым словосочетанием End Sub.

Описание функции:

```
Область видимости Function Имя функции _
        (Список параметров) As Тип результата
    Операторы
  Return Результат
   End Function
```

Область видимости задается одним из двух ключевых слов: Private или Public. Она определяет доступность функции из других модулей проекта. Функция, описанная как Private, доступна только в том модуле, в котором она описана. Функция, описанная как Public, доступна во всех модулях проекта.

Function – ключевое слово Visual Basic, показывающее, что данная подпрограмма является функцией.

Имя функции – уникальное название функции, построенное по правилу имен, которое отличает данную подпрограмму от всех других элементов проекта.

Список параметров – это перечисленные через запятую аргументы функции. Количество параметров может быть произвольным. Список параметров может отсутствовать. Круглые скобки в этом случае остаются пустыми, но все равно ставятся. Каждый аргумент в списке описывается отдельно по следующей схеме.

Режим передачи Имя параметра As Тип данных

Режим передачи параметра определяет способ передачи данных в подпрограмму. Существует два способа передачи: по ссылке и по значению.

Имя параметра – уникальное имя переменной, отличающее этот параметр от других параметров данной подпрограммы.

Тип данных показывает, к какому типу данных принадлежит данный параметр.

После списка параметров указывается Тип результата, который возвращает функция. Эта часть описания является обязательной. Даже если отсутствует список параметров, Тип результата указывать необходимо.

Начиная со следующей после заголовка функции строки, пишутся операторы подпрограммы, которые реализуют необходимую последовательность действий. Любая функция обязательно должна возвращать вычисленное значение в вызывающую программу. Это можно реализовать двумя способами.

1. С помощью оператора Return. После ключевого слова Return через пробел указывается возвращаемое значение. Этот оператор немедленно прекращает выполнение данной функции и возвращает управление в вызывающую программу.
2. Результат можно вернуть в вызывающую программу, используя имя функции. В этом случае среди операторов функции должен быть следующий оператор присваивания.

Имя функции = Возвращаемый результат

Заметим, что данный оператор присваивания в отличие от оператора Return не приводит к немедленному завершению функции.

Среди операторов, входящих в состав функции, может встречаться один особый оператор: Exit Function. Он немедленно прекращает выполнение данной функции, но не возвращает результат в вызывающую программу.

Описание функции заканчивается ключевым словосочетанием End Function.

1.6 Ветвления, множественный выбор

Разветвляющийся алгоритм - это алгоритм, который содержит одно или несколько логических условий и имеет 2 ветви вычислений. В программировании ветвление реализуется с помощью условного оператора, а условие выбора записывается в форме логического (условного) выражения. Логическое (условное) выражение может быть представлено в четырех видах: логическая константа, логическая переменная, простое условие или сложное условие. Любое логическое выражение может иметь только одно из двух значений: Истина (True) или Ложь (False).

Простое условие – это два выражения, между которыми стоит знак операции сравнения. В роли выражений могут выступать числа, числовые переменные, математические функции, арифметические выражения, строки, строковые переменные, строковые функции и строковые выражения. Оба выражения, участвующие в сравнении, обязательно должны принадлежать к одному и тому же типу. Если простое условие выполняется, оно имеет значение Истина (True). В противном случае оно имеет значение Ложь (False).

Сложное условие – это последовательность простых условий, логических переменных и логических констант, которые соединены между собой знаками логических операций. Традиционно каждую составную часть сложного условия берут в круглые скобки, хотя в некоторых языках программирования, Visual Basic, это является необязательным.

Условные операторы

1) Для программирования ветвящихся алгоритмов в Паскале существует условный оператор If. Он может принимать одну из форм:

If <условие> then <оператор1>
else<оператор2>;

или

If <условие> then <оператор>;

Оператор выполняется следующим образом. Сначала вычисляется выражение, записанное в условии. В результате его вычисление получается значение логического (булевского) типа. Если это значение – «истина», то выполняется оператор1, указанный после слова then. Если же в результате имеем «ложь», то выполняется оператор2. В случае, если вместо оператора1 или оператора2 следует серия операторов, то эту серию операторов необходимо заключить в операторные скобки begin…end.

Обратить внимание, что перед словом else точка с запятой не ставится.

2) Оператор if в языке C++ имеет следующую структуру:

if (условие) оператор_1; else оператор_2;

где условие - логическое выражение, переменная или константа.

Работает условный оператор следующим образом. Если условие оно не равно нулю, т.е. имеет значение истина (true), выполняется оператор_1. В противном случае, когда выражение равно нулю (ложь - false), то - оператор _2.

Если в операторе if требуется, чтобы в зависимости от значения условия выполнялся не один оператор, а несколько, то оператор if следует записывать в следующей форме:

```
if (условие)
{
оператор_1;
оператор_2;
...
}
else
{
оператор_1;
оператор_2;
...
}
```

Таким образом, транслятор воспринимает составной оператор как одно целое. В языке C++ предусмотрена сокращенная запись условного оператора. Она называется «условным выражением» или «тернарной операцией». Такое выражение выглядит так:

B1 ? B2 : B3

Сначала вычисляется значение выражения B1. Если оно отлично от нуля (истинно), то вычисляется значение выражения B2, которое и становится значением условного выражения. В противном случае вычисляется значение выражения B3, и оно становится значением условного выражения. Условное выражение удобно использовать в тех случаях, когда имеется некоторая переменная, которой можно присвоить одно из двух возможных значений. Типичным примером являются присваивание переменной значения большей из двух величин: max= (a>b) ?a:b;

3) Условный оператор в языке Бейсик (Visual Basic / VBA) имеет две формы синтаксиса: строчную (однострочный оператор) и блочную (многострочный оператор). Однострочный условный оператор имеет следующий синтаксис:

If Условное Выражение Then Оператор1 [Else Оператор2]

Многострочный условный оператор записывается в несколько строк. Причем распределение ключевых слов по строкам является обязательным и не может быть изменено. Синтаксис многострочного условного оператора:

If Условное Выражение Then

Группа Операторов 1
Else
Группа Операторов 2
End If

В отличие от однострочного условного оператора, ветви многострочного оператора могут содержать не одно, а несколько действий.

В языке Visual Basic существует еще один способ организации ветвления – функция IIf. Она имеет следующий синтаксис:

IIf(Условное выражение, Значение1, Значение2)

Функция проверяет истинность Условного Выражения. Если Условное Выражение имеет значение Истина (True), то функция IIf возвращает Значение1. Если Условное Выражение имеет значение Ложь (False), то функция IIf возвращает Значение2.

При решении сложных задач часто возникает ситуация, когда определенное действие (или набор действий) нужно выполнить после проверки не одного, а нескольких условий. В таких случаях используют конструкцию множественного условного перехода. В Visual Basic эта конструкция реализуется с помощью оператора множественного ветвления ElseIf, имеющего следующий синтаксис:

```
If Условное Выражение 1 Then
   Операторы 1
ElseIf Условное Выражение 2 Then
   Операторы 2
ElseIf Условное Выражение 3 Then
   Операторы 3
. . .
ElseIf Условное Выражение N Then
   Операторы N
Else
   Операторы Else
End If
```

После завершения работы оператора множественного ветвления выполнение программы продолжается с оператора, стоящего после ключевого словосочетания End If. Обратите внимание, что выполниться может только одна ветвь оператора, даже если истинными являются сразу несколько Условных Выражений. В таких случаях реализуется первая из подходящих ветвей оператора.

Начало каждой ветви условного оператора программы на языке Питон обозначается символом «:». Условие в операторе IF (если) записывается без скобок. Как таковое окончание оператора IF отсутствует. Python считает, что следующий оператор начинается в строке без отступа. Таким образом, в Python отступы играют важную роль.

Ключевое слово elif в Python является сокращением от else if (иначе если) и используется для организации вложенных условий (выбора).

Оператор выбора:

1) В Паскале оператор выбора Case организует переход на один из нескольких вариантов действий в зависимости от значения выражения, называемого селектором. Общий вид:

```
  Case k of
<const1>: <оператор1>;
<const2>: <оператор2>;
 …..
<constN>: <операторN>
else <операторN+1>
end;
```

Здесь k – выражение-селектор, которое может иметь только простой порядковый тип (целый, символьный, логический). <const1>, …<constN> - константы того же типа, что и селектор.

Оператор Case работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна значению селектора, то выполняется оператор, стоящий за словом else. Если же это слово отсутствует, то активизируется оператор, находящийся за границей Case, т.е. после слова end.

При использовании оператора Case должны выполняться следующие правила:

1. Выражение-селектор может иметь только простой порядковый тип (целый, символьный, логический).
2. Все константы, которые предшествуют операторам альтернатив, должны иметь тот же тип, что и селектор.
3. Все константы в альтернативах должны быть уникальны в пределах оператора выбора.

2) Оператор варианта switch в языке C++ необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы:

```
switch (выражение)
{
case значение_1: Операторы_1; break;
case значение_2: Операторы_2; break;
case значение_3: Операторы_3; break;
...
case значение_n: Операторы_n; break;
default: Операторы; break;
}
```

Оператор break необходим для того, чтобы осуществить выход из оператора switch. Если оператор break не указан, то будут выполняться следующие операторы из списка, несмотря на то, что значение, которым они помечены, не совпадает со значением выражения.

3) Оператор выбора применяется для программирования выбора одной из нескольких взаимоисключающих возможностей (альтернатив), при этом условием выбора ветви алгоритма является значение некоторого выражения. Оператор выбора в системе программирования Visual Basic имеет следующий синтаксис:

```
Select Case Выражение
   Case Список Значений 1
      Операторы 1
   Case Список Значений 2
      Операторы 2
   . . .
   Case Список Значений N
      Операторы N
   Case Else
      Операторы Else
End Select
```

Список значений – это последовательность выражений, разделенных запятым. Выражение может быть представлено в виде константы, числа, строки, переменной, арифметического, логического или строкового выражения. Также Список Значений может включать выражения специального вида: диапазон и луч.

Выражение-диапазон позволяет задать диапазон возможных значений. Оно имеет следующий синтаксис:

Выражение To Выражение

Например, диапазон 1 To 5 определяет отрезок от 1 до 5, включая границы. В Visual Basic границы всегда включаются в диапазон.

Выражение-луч используется, когда надо определить полуинтервал возможных значений. Оно имеет следующий синтаксис:

Is Знак операции сравнения Выражение

Например, луч Is > 10 задает множество чисел, превышающих 10.

1.7 Циклы

Если в программе возникает необходимость неоднократного выполнения некоторых операторов, то для этого используются операторы повтора (цикла). В языке Паскаль различают три вида операторов цикла: цикл с предусловием (while), цикл с постусловием (repeat) и цикл с параметром (for).

Если число требуемых повторений заранее известно, то используется оператор, называемый оператором цикла с параметром.

Оператор цикла с параметром имеет два варианта записи:

1) for <имя переменной> := <начальное значение> to <конечное значение> do

<тело цикла>

2) for <имя переменной> := <начальное значение> downto <конечное значение> do

<тело цикла>

Имя переменной – параметр цикла, простая переменная целого типа; <тело цикла> - операторы или оператор. Цикл повторяется до тех пор пока значение параметра лежит в интервале между начальным и конечным значения-ми. В первом варианте при каждом повторении цикла значение параметра увеличивается на 1, во втором - уменьшается на 1.

При первом обращении к оператору for вначале определяются начальное и конечное значения, и присваивается параметру цикла начальное значение. После этого циклически повторяются следующие действия.

1. Проверяется условие параметр цикла <= конечному значению.
2. Если условие выполнено, то оператор продолжает работу (выполняется оператор в теле цикла), если условие не выполнено, то оператор завершает работу и управление в программе передается на оператор, следующий за циклом.
3. Значение параметра изменяется (увеличивается на 1 или уменьшается на 1).

Если в теле цикла располагается более одного оператора, то они заключаются в операторные скобки begin … end;

Если число повторений заранее неизвестно, а задано лишь условие его повторения (или окончания), то используются операторы while и repeat. Оператор While часто называют оператором цикла с предусловием. Так как проверка условия выполнения цикла производится в самом начале оператора.

Общий вид: While <условие продолжения повторений> do
<тело цикла>;

Тело цикла – простой или составной оператор или операторы. Если операторов в теле цикла несколько, то тело цикла заключается в операторные скобки begin…end.

Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат – «истина», тело цикла выполняется и снова вычисляется выражение условия. Если результат–«ложь», происходят выход из цикла и переход к первому после while оператору.

Оператор цикла repeat аналогичен оператору while, но отличается от него, во-первых, тем, что условие проверяется после очередного выполнения операторов тела цикла и таким образом гарантируется хотя бы однократное выполнение цикла. Во-вторых, тем, что критерием прекращения цикла является равенство выражения константе true. За это данный оператор часто называют циклом с постусловием, так как он прекращает выполняться, как только условие, записанное после слова until, выполнится. Оператор цикла repeat состоит из заголовка, тела и условия окончания.

Общий вид: Repeat
<оператор>
. . . . .
<оператор>
until <условие окончания цикла>

В начале выполняется тело цикла, затем проверяется условие выхода из цикла. В любом случае, этот цикл выполняется хотя бы один раз. Если условие не выполняется, т.е. результатом выражения является False, то цикл активизируется еще раз. Если условие выполнено, то происходит выход из цикла. Использования операторных скобок, в случае, если тело цикла состоит из нескольких операторов, не требуется.

В C++ так же предусмотрены три оператора, реализующих циклический процесс: while, do-while и for.

В C++ существует оператор цикла с параметром следующей структуры:

```
for (начальные_присваивания; выражение; приращение) оператор;
или
for (начальные_присваивания; выражение; приращение)
{
оператор1;
оператор2;
...
}
```

Алгоритм работы цикла for следующий:

1. Выполняются начальные_присваивания.

2. Вычисляется значение выражения, если оно не равно 0 (true), то выполняется переход к п. 3. В противном случае выполнение цикла завершается.

3. Выполняется оператор.

4. Управление передается оператору приращение, после чего осуществляется переход к п. 2, то есть опять вычисляется значение выражения и т.д.

Оператор, реализующий цикл с предусловием в C++, имеет вид:

```
while (выражение) оператор;
или
while условие
{
оператор 1;
оператор 2;
...
оператор n;
}
```

В цикле с предусловием условие цикла располагается перед телом цикла.

Оператор, реализующий цикл с постусловием в C++, имеет вид:

```
do оператор while (выражение);
или
do
{
оператор_1;
```

оператор_2;
...
оператор_n;
}
while (выражение);

В Visual Basic цикл со счетчиком реализуется с помощью оператора For. Рассмотрим его синтаксис:

For Счетчик = Нач. значение To Кон. Значение Step Шаг
    Операторы тела цикла
Next

В синтаксической структуре принято выделять две части: заголовок цикла (первая строка оператора цикла) и тело цикла (блок операторов, стоящих между строками For и Next). В старых версиях языка Basic после ключевого слова Next необходимо было указывать Счетчик цикла. В Visual Basic это является необязательным.

Выполнение цикла со счетчиком происходит в несколько этапов.

1. Заголовок цикла проверяется на отсутствие противоречий. Это возможно в двух случаях.

• Если Начальное значение меньше Конечного Значения, то Шаг цикла должен быть больше нуля.

• Если Начальное значение больше Конечного Значения, то Шаг цикла должен быть меньше нуля.

Visual Basic позволяет не указывать Шаг в заголовке цикла, опуская при этом ключевое слово Step. В таких случаях Шаг цикла считается равным единице. Если заголовок цикла является противоречивым, то цикл выполняться не будет, а работа программы будет продолжена с оператора, стоящего после ключевого слова Next.

2. Если в заголовке цикла нет противоречий, то переменная Счетчик становится равной Начальному значению.

3. При данном значении Счетчика выполняются операторы тела цикла.

4. Значение счетчика изменяется на величину Шага. Если Шаг положительный, то значение Счетчика будет увеличиваться. Если Шаг отрицательный – уменьшаться. Если в заголовке цикла шаг не указан, то значение Счетчика будет увеличиваться на единицу.

5. Проверяется, попадает ли значение Счетчика в диапазон от Начального значения до Конечного значения. Если да, то происходит переход к пункту 3, и цикл выполняется еще раз. В противном случае работа цикла завершается.

6. Если среди операторов тела цикла встречается оператор Exit For, то выполнение цикла после этого оператора сразу прекращается независимо от значения Счетчика.

Синтаксис оператора цикла с предусловием в Visual Basic:

Do Условие Цикла

Тело цикла
Loop
Или полная версия оператора Do … Loop с предусловием:
Do [While | Until условие]
[инструкции]
[Exit Do]
[инструкции]
Loop
Оператор Do … Loop с предусловием организует проверку условия перед каждым входом в цикл и выполнение инструкций, входящих в тело цикла до тех пор, пока условие истинно, если оно записано после ключевого слова While, или ложно, если оно записано после ключевого слова Until.
Оператор Exit Do завершает выполнение цикла и передает управление оператору, следующему за инструкцией Loop.
При этом возможна ситуация, когда операторы тела цикла не выполнятся ни разу. Другими словами, условие цикла можно сформулировать таким образом, что управление никогда не попадет внутрь цикла.
Синтаксис оператора Do … Loop в Visual Basic с постусловием:
Do
[инструкции]
[Exit Do]
[инструкции]
Loop [While | Until условие]
Оператор Do … Loop с постусловием организует проверку условия после каждого выполнения тела цикла. Если условие, записанное после ключевого слова While, истинно, или условие, записанное после ключевого слова Until, ложно, то управление передается на первый оператор тела цикла, иначе выполняется оператор, следующий за инструкцией Loop.
Таким образом, в Visual Basic возможны четыре различных варианта цикла с условием.

- Do While Условное выражение

Тело цикла
Loop

- Do Until Условное Выражение

Тело цикла
Loop

- Do

Тело цикла
Loop While Условное выражение

- Do

Тело цикла
Loop Until Условное Выражение

Меняя Условные выражения, каждый вид цикла можно заменить на любой другой без потери работоспособности программы. Если Условие цик-

ла сформулировано с ошибкой, то программа может попасть в бесконечный цикл. Бесконечный цикл – это цикл, в котором количество повторов ничем не ограничено. В таких случаях говорят, что программа «зациклилась» или «зависла». Чаще всего бесконечный цикл возникает из-за ошибок в условии цикла: условие продолжения всегда имеет значение Истина (True) или условие завершения цикла всегда имеет значение Ложь (False). Среди операторов тела цикла может встречаться оператор Exit Do. Он прекращает выполнение цикла при любом значении его условия. Выполнение программы продолжается с оператора, стоящего сразу после ключевого слова Loop.

В Питоне синтаксис цикла с параметром выглядит следующим образом: for начальное_значение in range (конечное_значение−1) :

Поскольку диапазон чисел, формируемых функцией range(), начинается с 0, то верхней границей должно быть N−1.

Тело цикла начинается после символа «:», и все операторы тела цикла в Python должны иметь одинаковый отступ от начала строки. Как только отступ исчезает, Python считает, что тело цикла закончилось.

Циклы с заданным числом повторений не во всех системах программирования имеют возможность изменения параметра на заданный шаг. Поэтому можно использовать вложенные циклы с несколькими переменными – параметрами цикла, или применять цикл с предусловием.

Циклы с постусловием целесообразны в тех случаях, когда предыдущее значение выражения сравнивается в условии с текущим. Например, вычисление суммы бесконечного числового сходящегося ряда с заданной точностью.

Простейшая задача – посчитать сумму N первых чисел. Например, только четных чисел.

Циклы с параметром необходимы при работе с массивами как упорядоченными совокупностями однотипных данных.

В программах на Прологе повторяющиеся операции обычно выполняются при помощи правил, которые используют возврат и рекурсию. Существует четыре способа построения итеративных и рекурсивных правил:

- метод возврата после неудачи;
- метод отсечения и возврата;
- правило повтора, определяемое пользователем;
- обобщенное рекурсивное правило.

Правила повторений и рекурсии должны содержать средства управления их выполнением. Встроенные предикаты Турбо-Пролога fail и cut (!) используются для управления возвратами, а условия завершения используются для управления рекурсией. Правила Пролога, выполняющие повторения, используют возврат, а правила, выполняющие рекурсию, используют самовызов.

Возврат является автоматически инициируемым системой процессом, если не используются специальные средства управления им.

Метод возврата после неудачи может быть использован для управления вычислением внутренней цели при поиске всех возможных решений. Данный метод использует внутренний предикат Пролога fail.

Метод отсечения и возврата может быть использован для фильтрации данных, выбираемых из БД. Для управления возвратом при выборе данных по условию Пролог использует встроенный предикат отсечения, обозначаемый символом восклицательного знака (!). Этот предикат, вычисление которого всегда завершается успешно, заставляет внутренние подпрограммы унификации «забыть» все указатели возврата, установленные во время попыток вычислить текущую подцель. Другими словами, отсечение запрещает выполнение возврата ко всем альтернативным решениям текущей подцели. Однако, следующие подцели могут создать новые указатели возврата и тем самым создать условия поиска новых решений. Область действия данного предиката отсечения на них уже не распространяется.

Вид правила повторения, определяемого пользователем:

repeat.

repeat:- repeat.

Первый repeat является предложением, объявляющим предикат repeat истинным. Однако, поскольку имеется еще один вариант для данного предложения, то указатель возврата устанавливается на первый repeat. Второй repeat – это правило, которое использует само себя как компоненту (третий repeat). Второй repeat вызывает третий repeat, и этот вызов вычисляется успешно, так как первый repeat удовлетворяет подцели repeat. Предикат repeat будет вычисляться успешно при каждой новой попытке его вызвать после возврата. Факт будет использоваться для выполнения всех подцелей. Таким образом, repeat это рекурсивное правило, которое никогда не бывает неуспешным. Предикат repeat широко используется в качестве компоненты других правил, которая вызывает повторное выполнение всех следующих за ней компонент.

Правило, содержащее само себя в качестве компоненты, называется правилом рекурсии. Правила рекурсии, так же как правила повтора, реализуют повторное выполнение операций. Они весьма эффективны, например, при формировании запросов к базе данных, а также при обработке списков.

Во многих языках программирования можно написать рекурсивную функцию с помощью стандартных операторов цикла. В Python также есть понятие рекурсии как основы функционального программирования.

1.8 Массивы

Массив – это упорядоченная последовательность данных, состоящая из фиксированного числа элементов, имеющих один и тот же тип, и обозначаемая одним именем.

Название регулярный тип массивы получили за то, что в них объединены однородные элементы, упорядоченные (урегулированные) по индексам, определяющим положение каждого элементы в массиве.

Массиву присваивается имя, посредством которого можно ссылаться на него, как на единое целое. Элементы, образующие массив, упорядочены так, что каждому элементу соответствует совокупность номеров (индексов), определяющих его место в общей последовательности. Индексы представляют собой выражения простого типа. Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индекса нужного элемента. Описание массива определяет его имя, размер массива и тип данных.

Линейный (одномерный) массив – массив, у которого в описании задан только один индекс, если два индекса – то это двумерный массив и т.д. Одномерные массивы часто называют векторами, т.е. они представляют собой конечную последовательность пронумерованных элементов.

Присваивание начальных значений (заполнение массива) заключается в присваивании каждому элементу массива некоторого значения, заданного типа. Наиболее эффективно эта операция осуществляется при помощи оператора for. Ввод данных может осуществляться: с клавиатуры, при помощи различных формул, в том числе и датчика случайных чисел.

Индексированные элементы массива называются индексированными переменными и могут быть использованы так же, как и простые переменные. Например, они могут находиться в выражениях в качестве операндов, им можно присваивать любые значения, соответствующие их типу и т.д.

Алгоритм решения задач с использованием массивов:

- Описание массива
- Заполнение массива
- Вывод (распечатка) массива
- Выполнение условий задачи
- Вывод результата

Двумерный массив – структура данных, хранящая прямоугольную матрицу. В матрице каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен.

Общий вид описания массива в Паскале:

Type <имя нового типа данных>=array[<тип индекса>] of <тип компонентов>;

Далее, в перечне переменных указывается имя массива, и через двоеточие указывается имя нового типа данных. Массив может быть описан и без представления типа в разделе описания типов данных:

Var <имя массива>: array [<тип индекса>] of <тип компонентов>;

Чаще всего в качестве типа индекса используется интервальный целый тип. Для одномерных массивов индекс записывается так:

Var <имя массива>: array [начальное .. конечное] of <тип компонентов>;

В Паскале двумерный массив представляется массивом, элементами которого являются одномерные массивы:

Var <имя массива>: array [н1 .. к1, н2 .. к2] of <тип компонентов>;

В C++ одномерный массив описывают так:

тип имя_переменной [n];

где n - количество элементов в массиве, причем нумерация начинается с нуля: от 0 до n. Переменная n должна быть определена заранее как константа директивой #define.

Двумерный массив (матрицу) можно объявить так:

тип имя_переменной [n][m];

где n - количество строк (от 0 до n-1), m - количество столбцов (от 0 до m-1).

Обращаются к элементу матрицы, указывая последовательно в квадратных скобках соответствующие индексы:

Например, a[1][2] - элемент матрицы a, находящийся в первой строке и втором столбце.

Массиву, как и простой переменной, можно присвоить начальные значения в момент его описания.

Объявление массива в Visual Basic выполняется аналогично объявлению переменной. Для этого используются уже те же операторы Dim, Static, Public и Private. По способу описания массивы делятся на две основные группы:

- массив с заранее известным числом элементов;
- массив, число элементов которого заранее неизвестно.

При описании массива указывается его имя, затем ставятся круглые скобки, показывающие Visual Basic, что мы организуем массив, и задается тип данных, к которому будут принадлежать все элементы массива. Рассмотрим два случая объявления массива.

Dim a() As Integer

Эта конструкция описывает одномерный целочисленный массив a типа Integer, размер которого заранее неизвестен. Изначально в нем нет ни одного элемента. Размер этого массива будет определен позднее с помощью оператора ReDim.

Второй способ:

Dim name(n) as type

В предыдущих версиях Visual Basic массивы по способу описания делились на статические и динамические. Размер статического массива определялся один раз при его описании и не мог меняться в процессе выполнения программы. Размер динамического массива при описании не указывался, а задавался в процессе выполнения программы. Размер динамического массива можно было многократно менять в процессе выполнения программы. Причем допускалось как уменьшение размера массива, так и его увеличение. Начиная с версии Visual Basic 2005, понятие статического массива исчезает. Все массивы в Visual Basic 2005 являются динамическими, независимо от способа их объявления.

Оператор ReDim предназначен для изменения размера массива в процессе выполнения программы. Причем размер массива можно как умень-

шать, так и увеличивать. Но он не позволяет изменить тип элементов массива и размерность массива (количество используемых индексов). При изменении размера массива данные, хранящиеся в нем, могут теряться. Чтобы этого не происходило после слова ReDim необходимо поставить ключевое слово Preserve. В общем виде оператор ReDim записывается следующим образом

ReDim Имя массива(Номер последнего элемента)

или

ReDim Preserve Имя массива(Номер последнего элемента)

В языке Пролог вместо массивов существует понятие списка.

Список – это упорядоченный набор объектов одного и того же типа. Элементами списка могут быть целые числа, действительные числа, символы, строки, символические имена и структуры. Порядок расположения элементов в списке играет важную роль: те же самые элементы списка, упорядоченные иным способом, представляют уже совсем другой список.

Совокупность элементов списка заключается в квадратные скобки ([]), элементы друг от друга отделяются запятыми. Список может содержать произвольное число элементов, единственным ограничением является объем оперативной памяти. Количество элементов в списке называется его длиной. Список может содержать один элемент и даже не содержать ни одного элемента. Список, не содержащий элементов, называется пустым или нулевым списком.

Непустой список можно рассматривать как список, состоящий из двух частей: головы – первого элемента списка; и хвоста – остальной части списка. Голова является элементом списка, хвост является списком. Голова списка – это неделимое значение, хвост представляет собой список, составленный из того, что осталось от исходного списка в результате «отделения головы». Этот новый список обычно можно делить и дальше. Если список состоит из одного элемента, то его можно разделить на голову, которой будет этот самый элемент, и хвост, являющийся пустым списком. Пустой список нельзя разделить на голову и хвост.

Операция деления списка на голову и хвост обозначается при помощи вертикальной черты (|):

[Head | Tail].

В языке Питон существуют и массивы, и списки. Массивы имеют такой же синтаксис, как и в C++.

В языках Паскаль и Си списки являются динамическими структурами данных и неразрывно связаны с понятием указателя.

1.9 Структуры данных

Запись в языке Паскаль - это способ объединения нескольких переменных разных типов в одной. Благодаря этому достигается замечательная упорядоченность данных, программы при этом упрощаются и становятся логичнее.

Записи описываются в разделе type, который подобен разделам var или const. В этом разделе описываются все типы, определяемые пользователем.

Описываются записи с помощью служебного слова record, перед которым идет имя записи. После описываются все переменные, которые будут содержаться в записи, подобно тому, как они описываются в разделе var. Завершается запись словом end;.

После того, как вы опишете запись в разделе type, можно создать переменные нового типа в разделе var.

Существует два способа обращения к элементам записи:

1. Поля записи (ее внутренние переменные) могут быть изменены путем использования служебного слова with.

2. К полю записи можно обратиться, указав имя записи и через точку имя поля.

Структура в C++ является собранием одного или более объектов (переменных, массивов, указателей, других объектов), которые для удобства работы с ними объединены под одним именем.

Определение структуры состоит из двух шагов:

1. объявление структуры (задание нового типа данных определенного пользователем), структура состоит из полей; Синтаксис:

struct name

{..}

2. определение переменных типа структура;

Для обращения к полям структуры надо указать имя переменной и через точку - имя поля.

В классическом Бейсике структуры не предусмотрены. В Visual Basic ввели это понятие. Структура – это непустая совокупность нескольких элементов, каждый из которых может иметь свой тип данных. Причем Visual Basic допускает использование одной структуры внутри дугой. Отдельный элемент структуры называется полем. Помимо полей структура может содержать еще различные подпрограммы. Подпрограммы, входящие в состав структуры называются методами.

Описание структуры должно располагаться в самом начале модуля перед описанием всех подпрограмм, глобальных и локальных переменных. Как правило, описание структуры располагают сразу после слов Public Class. Описывать структуры внутри программы или подпрограммы нельзя.

При описании структуры необходимо перечислить все ее поля, указав для каждого из них тип данных. Затем надо описать все методы, входящие в состав структуры. В общем случае описание структуры выглядит следующим образом.

```
ОбластьВидимости Structure ИмяСтруктуры
   ОбластьВидимости ИмяПоля1 As ТипДанных
   ОбластьВидимости ИмяПоля2 As ТипДанных
   ...
   ОбластьВидимости ИмяПоляN As ТипДанных
```

```
    Описание методов
End Structure
```

Имена структуры, полей и методов должны строиться в соответствии с общим правилом имен. В качестве типов данных при описании полей структуры могут использоваться любые типы данных, существующие в Visual Basic, в том числе и ранее объявленные структуры.

Поле структуры может быть одномерным или многомерным массивом. В этом случае после имени поля ставятся круглые скобки. Если массив должен быть многомерным, то между скобками ставится необходимое количество запятых. Обратите внимание, что в составе структуры могут использоваться только массивы с неизвестным номером последнего элемента. Обработка массивов, являющихся полями структуры, ничем не отличается от обработки аналогичных массивов, описанных вне структуры.

В состав структуры мимо полей могут входить еще и методы. Метод – это подпрограмма, входящая в состав структуры и предназначенная для обработки полей только той структуры, где она описана. Поля других структур метод обрабатывать не может. Метод может быть как функцией, так и процедурой. В зависимости от этого несколько меняется описание метода.

Описание метода-процедуры.

```
ОбластьВидимости Sub ИмяПроцедуры(Параметры)
   Операторы
End Sub
```

Описание метода-функции.

```
ОбластьВидимости Function ИмяФункции(Параметры) _
            As ТипРезультата
   Операторы
   Return Результат
End Function
```

Область видимости метода определяет, будет ли данный метод доступен вне структуры. Public метод предназначен для обработки полей структуры и может быть вызван из любой другой подпрограммы, в том числе и не принадлежащей данной структуре. Private метод является вспомогательным и не может быть вызван извне структуры.

Для того чтобы обратиться к Public полю структуры или вызвать Public метод надо указать имя переменной, имеющей тип структуры, и через точку напечатать имя нужного поля или метода. Когда нужно обработать сразу несколько полей структуры, можно использовать оператор With. Он позволяет упростить текст программы, сделав его более наглядным. Синтаксис этого оператора:

```
With Имя переменной, имеющей структурный тип
  Операторы
End With
```

Внутри оператора With имя обрабатываемой переменной не указывается, а обращение к нужному полю или методу начинается с точки.

Кроме структур, в некоторых языках есть понятие множества. Например, в Pascal и Python.

Множества - это наборы однотипных логически связанных друг с другом объектов. Характер связей между объектами лишь подразумевается программистом и никак не контролируется Паскалем. Количество элементов, входящих во множество, может меняться в пределах от 0 до 256 (множество, не содержащее элементов, называется пустым). Именно непостоянством количества своих элементов множества отличаются от массивов и записей.

Два множества считаются эквивалентными тогда и только тогда, когда все их элементы одинаковы, причем порядок следования элементов в множестве безразличен. Если все элементы одного множества входят также и в другое, говорят о включении первого множества во второе. Пустое множество включается в любое другое.

Описание типа множества в Паскале имеет вид:

<имя типа> = SET OF <баз.тип>

Здесь <имя типа> - правильный идентификатор;

SET, OF - зарезервированные слова (множество, из);

<баз.тип> - базовый тип элементов множества, в качестве которого может использоваться любой порядковый тип, кроме WORD, INTEGER, LONGINT.

Для задания множества используется конструктор множества: список спецификаций элементов множества, отделяемых друг от друга запятыми; список обрамляется квадратными скобками. Спецификациями элементов могут быть константы или выражения базового типа, а также - тип-диапазон того же базового типа.

Над множествами определены следующие операции:

* пересечение множеств; результат содержит элементы, общие для обо-их множеств;

+ объединение множеств; результат содержит элементы первого множества, дополненные недостающими элементами из второго множества;

- разность множеств; результат содержит элементы из первого множества, которые не принадлежат второму;

= проверка эквивалентности; возвращает TRUE, если оба множества эквивалентны;

<> проверка неэквивалентности; возвращает TRUE, если оба множества неэквивалентны;

<= проверка вхождения; возвращает TRUE, если первое множество включено во второе;

>= проверка вхождения; возвращает TRUE, если второе множество включено в первое;

IN проверка принадлежности; в этой бинарной операции первый элемент - выражение, а второй - множество одного и того же типа; воз-вращает TRUE , если выражение имеет значение, принадлежащее множеству:

Дополнительно к этим операциям можно использовать две процедуры. INCLUDE - включает новый элемент во множество. Обращение к процедуре:

INCLUDE (S,I)

Здесь S - множество, состоящее из элементов базового типа TSetBase;

I - элемент типа TSetBase, который необходимо включить во множество.

EXCLUDE - исключает элемент из множества. Обращение:

EXCLUDE(S,I)

Параметры обращения - такие же, как у процедуры INCLUDE.

В Прологе, в отличие от некоторых императивных языков программирования, нет такой встроенной структуры данных, как множество. И, значит, нам придется реализовывать это понятие, опираясь на имеющиеся стандартные домены. В качестве базового домена используем стандартный списковый домен.

Под множеством мы будем понимать список, который не содержит повторных вхождений элементов. Другими словами, в нашем множестве любое значение не может встречаться более одного раза.

В Python определены такие структуры данных (составные типы) как последовательности и отображения (называемые также словарями). Словари позволяют устанавливать связи (ассоциации) «ключ-значение», поэтому с их помощью создаются так называемые ассоциативные массивы.

Последовательности, в свою очередь, подразделяются на изменяемые и неизменяемые. Под изменяемостью (изменчивостью) последовательности понимается возможность добавлять или удалять элементы этой последовательности (т.е. изменять количество элементов последовательности).

Для структур данных в Python определены функции (операции) и методы, принципиального различия между которыми нет, а есть различие синтаксическое (в правилах написания).

1.10 Файлы

Файлом называют именованную (то есть имеющую имя) совокупность данных, записанных на диске. Файл состоит из компонентов (элементов). При чтении или записи файловая переменная перемещается к очередному компоненту и делает его доступным для обработки.

Файлы делятся на:

1. Текстовые
2. Двоичные

Текстовый файл — файл, в котором каждый символ из используемого набора символов хранится в виде одного байта (кода, соответствующего символу). Текстовые файлы разбиваются на несколько строк с помощью специального символа «конец строки». Текстовый файл заканчивается специальным символом «конец строки».

В Паскале работа с файлами осуществляется через специальные типы. Это файловые типы, которые определяют тип файла, то есть фактически ука-

зывают его содержимое. С помощью этой переменной, которой присвоен необходимый тип, и осуществляется вся работа с файлами - открытие, запись, чтение, закрытие и т.п.

При работе с файлами существует определенный порядок действий, которого необходимо придерживаться:

1. Создание (описание) файловой переменной;
2. Связывание этой переменной с конкретным файлом на диске или с устройством ввода-вывода (экран, клавиатура, принтер и т.п.);
3. Открытие файла для записи либо чтения;
4. Действия с файлом: чтение либо запись;
5. Закрытие файла.

Возможно связать файловую переменную не только с физическим файлом на носителе информации, но и с устройством.

Последний этап - закрытие файла. В принципе, не обязательное условие для файлов, из которых мы читаем данные. Если не закроем - ошибки это не вызовет, последствий тоже. Однако обязательно закрывать файл, если мы осуществляли в него запись. Дело в том, что если мы пишем данные в файл на диске и забываем его закрыть, - информация не сохранится. Она (информация) помещается во временный буфер, который запишется на диск только при закрытии файла.

Типы файловых переменных

В Pascal имеется три типа переменных, которые определяют тип файла:

1. Text - текстовый файл. Из переменной такого типа мы сможем читать строки и символы.
2. File of _любой_тип_ - так называемые "типизированные" файлы, то есть файлы, имеющие тип. Этот тип определяет, какого рода информация содержится в файле и задается в переметре _любой_тип_.
3. File - нетипизированный файл. Когда мы указываем в качестве типа файла просто File, то есть без типа, то чтение и запись производятся путем указания количества байт, которые нужно прочитать, а также указанием области памяти, в которую нужно прочитать эти данные.

Связывание переменной с файлом выполняется одной и той же процедурой для всех типов файлов - Assign:

Assign(_переменная_файлового_типа_, 'путь к файлу');

В качестве параметров задаются переменная любого файлового типа и строка - путь к файлу, который, по правилам DOS, не может быть длиннее 79 символов.

Открытие файла заключается в использовании одной из трех имеющихся процедур:

1. Reset(_любая_файловая_переменная_);

Открывает файл на чтение. В качестве параметра - файловая переменная любого из перечисленных выше типов. Это может быть текстовый, типизированный либо не типизированный файл. В случае с текстовым файлом, он

открывается только на чтение. В случае с типизированным и нетипизированным файлом - он открывается на чтение и запись.

2. Append(T: Text);

Эта процедура открывает только текстовый файл на запись. То есть если вы используете текстовый файл и хотите производить в него запись, нужно использовать Append. Если чтение - Reset. В остальных случаях дело обходиться одной процедурой Reset.

Также обратите внимание, что если вы до этого уже открыли файл на чтение, вам не нужно закрывать его и открывать снова на запись. В этом случае файл закрывается сам и открывается заново. При записи данных в файл при открытии его с помощью этой процедуры они записываются в конец файла.

3. ReWrite(F) - создает новый файл либо перезаписывает существующий. Файл, открытый с помощью этой процедуры будет полностью перезаписан.

Способ проверки наличия файла на диске заключается в двух этапах: использовании ключей компилятора и функции IOResult, которая возвращает значение от только что выполненной операции ввода-вывода.

Ключи компилятора - это переключатели, которые контролируют ход выполнения программы исключая или включая реакцию на какие-нибудь условия.

Закрытие файла производится с помощью процедуры Close(F), где F - это переменная файлового типа. Эта процедура одна для всех типов файлов.

Запись и чтение файлов.

Чтение файлов. Чтение файлов производится с помощью процедур Read и Readln. Перед переменной, в которую помещается считанное значение, указывается переменная файлового типа (дескриптор файла):

Read(F, C);

Здесь F - дескриптор файла, C - переменная (Char, String - для текстовых, любого типа - для типизированных файлов).

Самой главной функции при чтении файлов является функция поверки на конец файла - Eof(F): Boolean;. В качестве параметра - файловая переменная любого типа. Функция возвращает TRUE если достигнут конец файла и FALSE иначе.

Запись в файлы. Запись в файлы производится с помощью процедур Write и Writeln. Как и в случае с чтением, перед записываемой в файл переменной указывается дескриптор файла:

Write(F, S);

Здесь F - дескриптор, S - переменная.

При этом переменная должна соответствовать типу файла.

Чтение из файлов без типа производится с помощью процедуры BlockRead.

BlockRead(F: File, Buf: Var, Size: Word, Result: Word)

F: File; - переменная типа File; Именно из этой переменной и происходит чтение данных.

Buf: Var; - переменная любого типа. В эту переменную помещаются прочитанные данные.

Size: Word; - количество считываемых байт.

Result: Word; - в эту переменную помещается реальное количество байт, которые были прочитаны.

Функция Sizeof - принимает в качестве параметра любую переменную и возвращает ее размер в байтах.

Дополнительный параметр процедуры Reset. Он указывает размер буфера, который используется для передачи данных. С текстовыми файлами он не используется.

В C++ операции с файлами можно осуществлять с помощью файловых указателей и с помощью потоков.

Работа с текстовыми файлами с помощью файловых указателей

Для записи данных в файл нужно выполнить:

1. Описать указатель на файл FILE *filename;

2. Открыть файл (функция fopen)

Описание функции

FILE *fopen(const *filename, const char *mode)

filename – строка, в которой хранится полное имя открываемого файла.

mode – строка, которая определяет режим работы с файлом; возможны следующие значения:

· «r» – открываем текстовый файл в режиме чтения;

· «w» – создаем текстовый файл;

· «a» – создаем или открываем текстовый файл для дозаписи в конец файла;

· «r+» – открываем текстовый файл в режиме чтения и записи;

· «w+» – открываем текстовый файл для исправления, старое содержимое выбрасывается;

· «a+» текстовый файл открывается или создается для исправления существующей информации и добавления новой в конец файла;

Функция возвращает указатель на файловую переменную или NULL при неудачном открытии файла.

3. Записать данных в файл (функция fprintf)

Функция fprintf аналогична функции printf, единственным отличием является первый параметр – указатель на файл. С помощью этой функции вывод осуществляется не на экран, а в файл.

4. Закрыть файл (функция fclose)

int fclose(FILE *filename);

Возвращает 0 при успешном закрытии файла и NULL в противном случае.

Кроме этих функций для работы с файлами есть еще две:

int remove(const char *filename);

Эта функция удаляет с диска файл, указатель на который хранится в файловой переменной filename. Функция возвращает ненулевое значение, если файл не удалось удалить.

int rename(const char *oldfilename, const char *newfilename);

Функция переименовывает файл; первый параметр – старое имя файла, второй – новое. Возвращает 0 при неудачном завершении программы.

Для чтения данных из файла нужно выполнить:

1. Описать указатель на файл FILE *filename;

2. Открыть файл (функция fopen)

3. Считать данные из файла (функция fscanf)

Функция fscanf аналогична функции scanf, единственным отличием является первый параметр – указатель на файл. С помощью этой функции вывод осуществляется не на экран, а в файл. В случае необходимости при чтении надо контролировать возможность чтения очередного компонента с помощью функции feof.

4. Закрыть файл (функция fclose).

Работа с текстовыми файлами с помощью файловых потоков

Для работы с файлами используются специальные типы данные, называемые потоками. Поток ifstream служит для работы с файлами в режиме чтения. Поток ofstream служит для работы с файлами в режиме записи. Для работы с файлами в режиме, как чтения, так и записи служит поток fstream.

В программах на C++ при работе с текстовыми файлами необходимо подключать библиотеки iostream и fstream.

Для того, чтобы записывать данные в текстовый файл необходимо:

1. Описать переменную типа ofstream

2. Отрыть файл с помощью функции open.

3. Вывести информацию в файл с помощью cout.

4. Обязательно закрыть файл.

Для того, чтобы считывать данные из текстового файл необходимо:

1. Описать переменную типа ifstream

2. Отрыть файл с помощью функции open.

3. Считать информацию из файла с помощью cin, при считывании каждой порции данных необходимо проверять, что чтение возможно.

4. Закрыть файл.

Запись информации в текстовый файл

Для того, чтобы начать работать с текстовым файлом необходимо описать переменную типа ofstream. Например, с помощью оператора

ofstream F;

будет создана переменная F для записи информации в файл.

На следующем этапе файл необходимо открыть для записи. В общем случае оператор открытия файла будет иметь вид:

F.open("file", mode);

Здесь F – переменная, описанная как ofstream,

file – полное имя файла на диске,

mode – режим работы с открываемым файлом:

· ios::in – открыть файл в режиме чтения данных, этот режим является режимом по умолчанию для потоков ifstream;

· ios::out – открыть файл в режиме записи данных, этот режим является режимом по умолчанию для потоков ofstream;

· ios::app – открыть файл в режиме записи данных в конец файла;

· ios::ate – передвинуться в конец уже открытого файла;

· ios::trunc – очистить файл, это же происходит в режиме ios::out;

· ios::nocreate – не выполнять операцию открытия файл, если он не существует;

· ios::noreplace – не открывать существующий файл.

· ios::binary – открыть двоичный файл

Параметр mode может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока

ios::in – для потоков ifstream,

ios::out – для потоков ofstream.

После удачного открытия файла (в любом режиме) в переменной F будет храниться true, в противном случае false. Это позволит проверять корректность операции открытия файла.

Открыть файл в режиме записи можно одним из следующих способов:

1) ofstream F;

2) режим ios::out является режимом по умолчанию для потока ofstream. ofstream F;

3) объединяет описание переменной типа поток и открытие файла в одном операторе.

После открытия файла в режиме записи, будет создан пустой файл, в который можно будет записывать информацию. Если Вы хотите открыть существующий файл, то в качестве режима следует использовать значение ios::app.

После открытия файла в режиме записи, в него можно писать точно также, как и на экран, только вместо стандартного устройства вывода cout необходимо указать имя открытого для записи файла.

Для того чтобы прочитать информацию из текстового файла необходимо описать переменную типа ifstream. После этого необходимо открыть для чтения с помощью оператора open.

После открытия файла в режиме чтения, из него можно считывать информацию точно так же, как и клавиатуры, только вместо стандартного устройства ввода cin необходимо указать имя открытого для чтения файла.

Для проверки достигнут или нет конец файла, служит функция F.eof()

Здесь F – имя потока, функция возвращает логическое значение: true – если достигнут конец файла, если не достигнут функция возвращает значение false.

Обработка двоичных файлов в С++

При записи в двоичный файл символы и числа записываются в виде последовательности байт (в своем внутреннем двоичном представлении в памяти компьютера).

Порядок работы с двоичными и тестовыми файлами аналогичен.

Для того, чтобы записать данные двоичный файл необходимо:

1. Описать переменную типа FILE* с помощью оператора FILE *filename;

filename – имя переменной, где будет храниться указатель на файл.

2. Отрыть файл с помощью функции fopen.

3. Записать информацию в файл с помощью функции fwrite.

4. Закрыть файл с помощью функции fclose.

Двоичный файл – последовательная структура данных, после открытия файла доступен первый байт. Можно последовательно считывать или записывать данные из файла.

Пользовательские файлы в Прологе описываются в разделе описания доменов следующим образом:

file = <символическое имя файла1>;...;
<символическое имя файлаN>

При описании файловых доменов тип домена file располагается слева от равенства, а символические имена файлов — справа. Их еще называют внутренними или логическими именами файлов, в отличие от внешних или физических имен файлов. Символическое имя файла должно начинаться со строчной буквы.

Кроме пользовательских файлов, имеются стандартные файлы (или устройства), которые не нужно описывать в разделе описания доменов. Это:

- stdin(стандартное устройство ввода);
- stdout(стандартное устройство вывода);
- stderror(стандартное устройство вывода сообщений об ошибках);
- keyboard(клавиатура);
- screen(монитор);
- printer(параллельный порт принтера);
- coml(последовательный порт).

По умолчанию стандартным устройством ввода является клавиатура, а стандартным устройством вывода — монитор. Чтобы начать работу с пользовательским файлом, его нужно открыть, а по завершении работы— закрыть. Стандартные устройства ввода/вывода, а также параллельный и последовательный порты открывать и закрывать не нужно.

Встроенные предикаты, предназначенные для открытия файлов.

Каждый из следующих четырех предикатов имеет два входных параметра. Первый параметр — это внутреннее символическое имя, указанное в разделе описания доменов, второй параметр — это строка, представляющая внешнее имя файла.

Предикат openread открывает файл только для чтения. Если файл с указанным внешним именем не будет обнаружен, предикат терпит неудачу и выводит соответствующее сообщение об ошибке.

Предикат openwrite открывает файл только для записи. Этот предикат создает на диске новый файл. Если файл с указанным внешним именем уже существует, он будет стерт. Если по какой-то причине файл не может быть создан, предикат терпит неудачу и выводит соответствующее сообщение об ошибке.

Предикат openappend открывает файл только для дозаписи в конец файла. Если файл с указанным именем не будет обнаружен, предикат выводит соответствующее сообщение об ошибке.

Предикат openmodify открывает файл для чтения и записи одновременно. Если файл с указанным именем не будет обнаружен, предикат выводит соответствующее сообщение об ошибке.

Для того чтобы проверить, существует ли файл с указанным именем в указанном месте, используется предикат existfile. Этот предикат имеет один аргумент. Предикат истинен, если файл с именем, указанным в качестве его единственного параметра, существует, и ложен — в противном случае.

Эти предикаты связывают символическое имя файла с физическим именем открываемого файла. Поэтому, в отличие от других языков программирования, например, Паскаля, нам нет необходимости перед операцией открытия файла проводить операцию связывания внутреннего и внешнего имен файла.

Поскольку символ "\", обычно используемый для разделения имен каталогов, применяется в Турбо Прологе для записи кодов символов, требуется использовать вместо одного обратного слэша два ("\\").

Файлы в Прологе могут обрабатываться только последовательно.

Каждый запрос на чтение из входного файла приводит к чтению в текущей позиции текущего входного потока. После этого текущая позиция будет перемещена на следующий, еще не прочитанный элемент данных. Следующий запрос на чтение приведет к считыванию, начиная с этой новой текущей позиции.

Запись производится точно так же: каждый запрос на вывод информации приведет к тому, что она будет присоединена к концу текущего выходного потока.

Для работы с текстовым файлом в Питоне прежде всего нужно создать специальный объект – дескриптор файла, а потом использовать методы этого дескриптора для чтения и записи данных. При создании дескриптора нужно указать строку с именем файла и строку с описанием варианта доступа. Вариантов доступа бывает три: ’ r ’ (только чтение), ’w’ (запись) и ’a’ (дополнение).

Чтение возможно только из существующего файла. Если при открытии на запись или дополнение указано имя несуществующего файла, он будет создан.

Функция open() служит для открытия файла по имени для чтения, записи или изменения. Для прекращения работы с файлом (высвобождения дескриптора) используется метод close ().

Метод readline () читает из файла строку, а при повторном использовании - следующую строку. Метод readlines () читает из файла все строки и формирует из них список.

1.11 Графика.

Экран дисплея ПК представляет собой прямоугольное поле, состоящее из большого количества точек. Дисплей может работать в текстовом и графическом режимах. Но в отличие от текстового режима в графическом режиме имеется возможность изменять цвет каждой точки.

Чтобы сделать процесс графического программирования более эффективным, фирма Borland International разработала специализированную библиотеку Graph для системы Turbo Pascal (в этом библиотечном модуле содержится 79 графических процедур, функций, различных стандартных констант и типов данных), набор драйверов, позволяющих работать с разными типами мониторов, и набор шрифтов для вывода на графический экран текстов разной величины и формы. В современных средах программирования, например, PascalABC, этот модуль не действует, поэтому там были разработаны другие средства манипулирования графическим режимом. Однако, для удобства программистов часть функций имеет схожий синтаксис.

Для рисования в Pascal ABC необходимо запустить специальный модуль GraphABC, использование специальных функций и процедур помогут нарисовать точку, отрезок, окружность и прямоугольник и другие фигуры:

Первой инструкцией программы должна быть инструкция uses GraphABC;

SetPixel(x,y,color) - Закрашивает один пиксел с координатами (x,y) цветом color

LineTo(x,y) - рисует отрезок от текущего положения пера до точки (x,y); координаты пера при этом также становятся равными (x,y).

Line(x1,y1,x2,y2) - рисует отрезок с началом в точке (x1,y1) и концом в точке (x2,y2).

SetPenColor(color) - устанавливает цвет пера, задаваемый параметром color.

SetPenWidth(n) - устанавливает ширину (толщину) пера, равную n пикселям.

Rectangle(x1,y1,x2,y2) - рисует прямоугольник, заданный координатами противоположных вершин (x1,y1) и (x2,y2).

FloodFill(x,y,color) - заливает область одного цвета цветом color, начиная с точки (x,y).

SetBrushColor(color) - устанавливает цвет кисти. Заливка кистью распространяется на замкнутый контур, описание которого следует за процедурой установки цвета кисти.

Ellipse(x1,y1,x2,y2) - рисует эллипс, заданный своим описанным прямоугольником с координатами противоположных вершин (x1,y1) и (x2,y2).

Circle(x,y,r) - рисует окружность с центром в точке (x,y) и радиусом r.

Arc(x,y,r,a1,a2) - Рисует дугу окружности с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки).

Цвета в PascalABC:

clBlack – черный
clPurple – фиолетовый
clWhite – белый
clMaroon – темно-красный
clRed – красный
clNavy – темно-синий
clGreen – зеленый
clBrown – коричневый
clBlue – синий
clSkyBlue – голубой
clYellow – желтый
clCream – кремовый
clAqua – бирюзовый
clOlive – оливковый
clFuchsia – сиреневый
clTeal – сине-зеленый
clGray – темно-серый
clLime – ярко-зеленый
clMoneyGreen – цвет зеленых денег
clLtGray – светло-серый
clDkGray – темно-серый
clMedGray – серый
clSilver – серебряный

Цветовые возможности функций графической библиотеки Borland C описываются в терминах цветовой палитры режима – закона, по которому каждому допустимому значению атрибута пикселя ставится в соответствие цвет, которым этот пиксель будет отображаться на экране.

Палитру режима можно представить как таблицу, содержащую столько строк (входов), сколько значений допускается для атрибута пикселя в данном графическом режиме. Строки палитры режима нумеруются от нуля до N_palette-1. В строке с номером k содержится код цвета, которым аппаратура видеоадаптера отображает на экране все пиксели страницы, атрибуты которых равны k. Нулевой вход палитры режима, кроме того, определяет цвет фона экрана.

Все графические режимы можно разделить на три группы:

- монохромные режимы, в которых все пиксели могут быть двух цветов – основного и фонового. Палитру таких режимов изменить невозможно;

- цветные режимы с фиксированной палитрой. Для изменения палитры режима нужно менять графический режим, что приводит к потере содержимого видеопамяти;

- графические режимы, позволяющие динамически (без потери содержимого видеопамяти) изменять код цвета по любому входу палитры режима.

Для разработчиков на языке Си в системах объектно-ориентированного программирования была написана обширная библиотека windows.h, позволявшая в значительной степени унифицировать процесс разработки. Всю спецификацию программирования под Windows назвали API (Application Programming Interface) - интерфейс прикладного программирования. API покрывает собой всё: графику, элементы управления, системные функции, работу с устройствами и ещё многое другое. По сути, API функции - это промежуточное звено между пользователем и ядром операционной системы.

Благодаря API, в Visual C++ имеются методы для рисования: прямоугольника (Rectangle); эллипса (Ellipse); скругленного прямоугольника (RoundRect); сегмента эллипса (Chord); сектора эллипса (Pie); замкнутого многоугольника; составного замкнутого многоугольника.

Для рисования фигур важен атрибут контекста устройства, называемый текущей кистью. Он задает кисть как объект GDI, с помощью которого производится закрашивание внутренней области фигуры. Текущая кисть устанавливается методомSelectObject или SelectStockObject, если в качестве параметра передать одну из констант: BLACK_BRUSH, DKGRAY_BRUSH, GRAY_BRUSH, HOLLOW_BRUSH, LTGRAY_BRUSH, NULL_BRUSH, WHITE_BRUSH.

Python может работать с несколькими графическими библиотеками, обеспечивая создание сложных приложений с развитым графическим пользовательским интерфейсом.

Команды, обеспечивающие рисование, по принципу «черепашки» в языке Питон аналогичны тем, которые содержались в операторе DRAW в старых версиях Бейсика. В настоящее время некоторые интерпретаторы поддерживают данный режим, например, Microsoft Small Basic.

Этот принцип можно реализовать и в логических языках программирования.

Основное предназначение модуля Tkinter - создание графических интерфейсов (GUI - Graphical User Interface) для программ на Python. Но благодаря наличию элемента графического интерфейса canvas Tkinter можно применять для рисования на основании координат, рассчитанных по формулам, используя доступные элементы векторной графики: кривые, дуги, эллипсы, прямоугольники и пр.

Глава 2. Визуальные среды программирования

2.1. Формы, проекты

Большинство приложений для операционной системы Windows имеют оконный режим. Поэтому результат работы в визуальных средах (Delphi, Microsoft Visual Studio т.д.) – это не один исполняемый файл, а проект, состоящий из нескольких частей:

1) экранная форма с основными компонентами;
2) дополнительные (дочерние или равноправные формы);
3) код на соответствующем языке программирования;
4) связующий файл проекта;
5) дополнительные (служебные) файлы и библиотеки.

Таблица расширений файлов проектов

Система ООП	Форма	Код	Проект
Delphi	.dfm	.pas	.dpr
Lazarus	.lfm	.pas	.lpr
PascalABC.Net	.fmabc	.pas	.pabcproj
MS VB	.vb	.vb	.vbproj
MS VC#	.cs	.cs	.csproj
Visual Prolog	.cl	.pro	.prj6

Окно конструктора форм является основным рабочим окном, в котором выполняется визуальное проектирование интерфейса приложения. В окне форм визуально создаются все формы приложения. В начале работы экранная форма пуста. В процессе проектирования интерфейса на ней располагаются различные элементы управления.

По умолчанию форма имеет классический вид, однако, в свойствах объектов можно менять основные параметры: размеры, положение на экране, количество кнопок и т.д.

2.2. Основные компоненты

Компоненты, созданные для систем Microsoft и .Net, имеют почти одинаковые названия и назначение. Те же по смыслу компоненты в Delphi и Lazarus немного различаются по синтаксису. В одних средах компоненты в списке перечисляются по алфавиту, в других – разнесены по вкладкам. Свойства компонент перечисляются в инспекторе объектов.

Стандартные компоненты:

1) кнопка (button) и ее разновидности с предустановленными изображениями и кодом (например, ОК, Отмена и т.д);
2) ввод-вывод текста (label, memo, edit (textbox), richedit и т.д.) – позволяют вводить символьные данные и выводить результат или пояснения в символьном виде;
3) флажки и переключатели (checkbox, radiobutton);
4) работа со списками (listbox, combobox) – выбор данных;
5) основное и контекстное меню;
6) полосы прокрутки и группировка объектов;

7) поле для рисования фигур и вывода графики.
Дополнительные компоненты:
1) диалоговые окна и файлы;
2) список, таблица;
3) графика и мультимедиа.

Глава 3. Решение прикладных задач

Рассмотрим решение задач в системах программирования Small Basic, Visual Basic (VBA), PascalABC (Delphi, Lazarus), DevC++ (Microsoft Visual C++, Visual C#), TurboProlog (VisualProlog), Python. В визуальных средах можно использовать консольный режим работы или диалоговые окна ввода-вывода и обработчик кнопки основного кода.

3.1. Линейные программы

1.1. Вычислить значение функции $y = \frac{sin^2 x-4}{|x+lnx|}$ при заданном значении x>0.

PascalABC:	Dev-C++ (стандартная библиотека):	MS Visual C++ (потоковая библиотека):

program zadacha1;
var x,y:real;
begin
readln(x);
y:=(sqr(sin(x))-
4)/abs(x+ln(x));
writeln(y:3:3);
end.
Контрольный счет: x=2
y=-1,178
	#include <stdio.h>
	#include <math.h>
	int main()
	{
	float x,y;
	scanf("%f",&x);
	y=(sin(x)*sin(x)-4)
	/fabs(x+log(x));
	printf("%3.3f",y);
	return 0;
	}
	Контрольный счет: x=2
	y=-1,178
		#include "stdafx.h"
		#include <iostream>
		#include <math.h>
		using namespace std;
		int _tmain(int argc,
		_TCHAR* argv[])
		{
		float x;
		cin>>x;
		float y=(sin(x)*sin(x)-4)/
		fabs(x+log(x));
		cout.precision(4);
		cout<<y;
		return 0;
		}
		Контрольный счет: x=2
		y=-1,178

Small Basic:	VBA (Microsoft Excel):	Python:
'zadacha 1 input x y=(sin(x)^2- 4)/abs(x+log(x)) y=round(y,3) print y end Контрольный счет: x=2 y=-1,178	Sub zadacha1() Dim x, y As Double Sheets("Лист1").Activate x = Cells(1, 1).Value y = (Sin(x) ^ 2 - 4) / Abs(x + Log(x)) Cells(1, 2).Value = y End Sub Контрольный счет: a1=2 b1=-1,178	# -*- coding: utf-8 -*- from math import sin, fabs, log x=float(input()) y=(sin(x)**2-4) /fabs(x+log(x)) print(round(y,3)) Контрольный счет: x=2 y=-1,178
Visual Prolog (консоль):		Turbo Prolog
implement main open core, console		predicates igrek(real, real)

class predicates igrek:(real, real) procedure(i,o). clauses igrek(X,Y) :- Y=(math::sin(X)*math:: sin(X)-4)/math:: abs(X+math::ln(2)). run():- console::init(),X=read(), igrek(X,Y),write(Y),nl. end implement main goal mainExe::run(main::run). Контрольный счет: X=2, Y=-1,178…	clauses igrek(X,Y):- Y=(sin(X)*sin(X)- 4)/abs(X+ln(X)), write(Y), nl. goal igrek(2,Y). Контрольный счет: X=2, Y=-1,178…

3.2. Разветвляющиеся программы.

Задача: Решить квадратное уравнение. Найти действительные корни.

PascalABC:	Dev-C++ (потоковая библиотека):	C++ (стандартная библиотека):

Program zadacha2;
Var A,B,C,D:Integer;
X,X1,X2:Real;
Begin
Write('Задайте A=');
Read(A);
Write('Задайте B=');
Read(B);
Write('Задайте C=');
Read(C);
D:=Sqr(B)-4*A*C;
If D<0 Then
Writeln('Решений Нет')
Else
Begin
If D=0 Then
Begin
X:=-B/(2*A);
Writeln(X);
End
Else
Begin
X1:=(-B-Sqrt(D))/(2*A);
X2:=(-B+Sqrt(D))/(2*A);
Writeln('X1=',X1);
Writeln('X2=',X2);
	#include <iostream>
	#include <math.h>
	using namespace std;
	int main()
	{float a,b,c,d,x1,x2;
	//Ввод значений коэффициентов
	cout<<"a=";cin>>a;
	cout<<"b=";cin>>b;
	cout<<"c=";cin>>c;
	//дискриминант
	d=b*b-4*a*c;
	//Если дискриминант отрицателен,
	if (d<0)
	cout<<"Real roots are not present";
	else
	{
	x1=(-b+sqrt(d))/2/a;
	x2=(-b-sqrt(d))/(2*a);
	cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
	}
	return 0;
	}
		#include <stdio.h>
		#include <stdlib.h>
		#include <math.h>
		Void main()
		{
		double a, b, c, d, x1, x2;
		printf("Напишите значение a: ");
		scanf("%lf", &a);
		if(a == 0){
		printf("Первый коэффициент квадраного уравнения не может быть равен 0\n");
		exit(0);
		}
		printf("Напишите значение b: ");
		scanf("%lf", &b);
		printf("Напишите значение c: ");
		scanf("%lf", &c);
		d = b * b - 4* a * c;
		if(d < 0){
		printf("Уравнение не имеет действительных

End;
End;
Readln;
End.
Контрольный счет:
a=1, b=2, c=-3
x1=1, x2=-3
	Контрольный счет:
	a=1, b=2, c=-3
	x1=1, x2=-3
		решений\n");
		exit(0);
		}
		x1 =(-b - sqrt(d))/(2 * a);
		x2 =(-b + sqrt(d))/(2 * a);
		printf("Ответ: x1 = %g, x2 = %g\n", x1, x2);
		}

Small Basic:	VBA (Microsoft Excel):	Python:

PRINT " Введите коэффициенты a, b, c : " ;
INPUT a, b, c
IF (a = 0) AND (b = 0) AND (c = 0) THEN
PRINT " x - любое число"
ELSE
IF (a = 0) AND (b <> 0) THEN
PRINT "Линейное уравнение, корень один : x = "; -c / b
ELSE
IF (a = 0) AND (b = 0) AND (c <> 0) THEN
PRINT "Неправильное уравнение."
ELSE
Discr = b * b - 4 * a * c
IF Discr > 0 THEN
x1 = (-b + SQR(Discr)) / (2 * a)
x2 = (-b - SQR(Discr)) / (2 * a)
PRINT "x1 = "; x1; "; x2 = "; x2
ELSE
IF Discr = 0 THEN
x1 = - b / (2 * a)
PRINT "Корни: x1 = "; x1; "; x2 = "; x1
ELSE PRINT
	Sub www()
	Dim a#, b#, c#, d#
	a = Val(InputBox("a =""))
	b = Val(InputBox("b =""))
	c = Val(InputBox("c =""))
	If (a Or b Or c) = 0 Then
	MsgBox "Решение не определено"
	ElseIf (a Or b) = 0 Then
	MsgBox "Неверные исходные данные"
	ElseIf a = 0 Then
	MsgBox "x = " & -c / b
	Else
	d = b ^ 2 - 4 * a * c
	If d < 0 Then
	MsgBox "Нет действительных корней"
	ElseIf d > 0 Then
	MsgBox "x1 = " & (-b - Sqr(d)) / 2 / a & ", x2 = " & (Sqr(d) - b) / 2 / a
	Else
	MsgBox "x12 = " & -b / 2 / a
	End If
	End If
	End Sub
		print("Введите коэффициенты для квадратного уравнения (ax^2 + bx + c = 0):")
		a = float(input("a = "))
		b = float(input("b = "))
		c = float(input("c = "))
		discr = b**2 - 4 * a * c;
		print("Дискриминант D = %.2f" % discr)
		if discr > 0:
		import math
		x1 = (-b + math.sqrt(discr)) / (2 * a)
		x2 = (-b - math.sqrt(discr)) / (2 * a)
		print("x1 = %.2f \nx2 = %.2f" % (x1, x2))
		elif discr == 0:
		x = -b / (2 * a)
		print("x = %.2f" % x)
		else:
		print("Корней нет")

"Действительных кор-ней нет." END IF END IF END IF END IF END IF : PRINT		
Visual Prolog (консоль):		Turbo Prolog
implement main open core, console, math class predicates solve:(real, real, real). clauses run():- console::init(), write("введите значения переменных"), nl, write("введите значение a"), nl, A=read(), write("введите значение b"), nl, B=read(), write("введите значение c"), nl, C=read(), solve(A,B,C). solve(A,B,C):- A = 0,!, write("A=0 => X="), X = -C/B, write(X). solve(A,B,C):-D=B*B-4*A*C, D < 0, !,write("D<0 => корней нет"). solve(A,B,C):-D=B*B-4*A*C,D = 0,!, write("D=0 => X1=X2="), X =-B/(2*A), write(X). solve(A,B,C):-D=B*B-4*A*C,X1 =(-B+sqrt(D))/(2*A) ,X2 =(-B-sqrt(D))/(2*A), write("D>0 => X1="), write(X1), write(" X2="), write(X2). end implement main goal mainExe::run(main::run). Контрольный счет: a=1, b=2, c=-3 x1=1, x2=-3		predicates replay(real,real,real). clauses replay(A,B,C):- write(A,"x*x+(",B,")x+",C,"=0"),nl,X=(-B+sqrt(B*B-4*A*C))/(2*A),Y=(-B-sqrt(B*B-4*A*C))/(2*A),write("x1=",X),nl,write("x2=",Y),nl. Goal: replay(1,2,-3). Контрольный счет: a=1, b=2, c=-3 x1=1, x2=-3

3.3. Циклические программы

Найти сумму 10 первых чисел.

PascalABC:	Dev-C++ (потоковая библиотека, цикл-пока):	Dev-C++ (потоковая библиотека, цикл-для):
Program zadacha3; var i,s:integer;	#include <iostream> using namespace std;	#include <iostream> using namespace std;

begin;
s:=0;
FOR i := 1 TO 10 do
s:=s+i;
writeln(s);
end.
Контрольный счет: 55
	int main()
	{
	unsigned int i,S;
	S=0;
	i=1;
	while (i<=10)
	{
	S+=i;
	i+=1;
	}
	cout<<"S="<<S<<"\n";
	return 0;
	}
	Контрольный счет: 55
		int main()
		{
		unsigned int i,S;
		S=0;
		for (i=1;i<=10;i++)
		S+=i;
		cout<<"S="<<S<<"\n";
		return 0;
		}
		Контрольный счет: 55

Small Basic:	Visual Prolog (консоль):	Python:
S=0 FOR I=1 TO 10 S=S+I NEXT I PRINT "Сумма ="; S Контрольный счет: 55	implement main open core, console class predicates sum:(integer, integer, integer) procedure (i, i, o). clauses sum(A, B, 0) :- A > B, !. sum(A, B, S + A) :- sum(A + 2, B, S), !. sum(A, B, S) :- sum(A + 1, B, S). clauses run() :- init(), sum(-1, 10, S), write(S), _ = readChar(). end implement main goal mainExe::run(main::run). Контрольный счет: 55	# -*- coding: utf-8 -*- S=0 for i in range (11) : S=S+i print ('Результат : ' ,S) Контрольный счет: 55

3.4. Одномерные массивы

Найти наибольший элемент массива.

PascalABC:	Dev-C++ (потоковая библиотека):	Turbo Prolog:
var a:array[1..10] of inte-	#include<iostream>	Domains

ger;
max:integer;
i:integer;
begin
writeln('введите 10 элементов массива');
max:=-(MAXINT+1);
for i:=1 to 10 do begin
readln(a[i]);
if max<a[i] then max:=a[i];
end;
writeln('Максимальный элемент массива = ', max);
end.
	using namespace std;
	int main()
	{
	float x[10];
	int i,n;
	float max;
	cout<<"\n N=";
	cin>>n;
	cout<<"\n Vvedite massiv X \n";
	for(i=0;i<n;i++)
	cin>>x[i];
	for(max=x[0],i=1;i<n;i++)
	if (x[i]>max)
	max=x[i];
	cout<<max;
	return 0;
	}
		intlist = integer*
		Predicates
		max(integer,integer,integer)
		maxlist(intlist, integer)
		Clauses
		max(X,Y,X):- X>=Y.
		max(X,Y,Y):- X<Y.
		maxlist([X],X).
		maxlist([X|T],MAX):-
		maxlist(T,MAXT),
		max(X,MAXT,MAX).

Small Basic:	VBA (Microsoft Excel):	Python:

INPUT "N = "; N
DIM A(N)
FOR i = 1 TO N
PRINT "A("; i; ") = ";
INPUT A(i)
NEXT i
Amax = A(1)
FOR i = 2 TO N
IF A(i) > Amax THEN Amax = A(i)
NEXT i
PRINT "Наибольший элемент" ; Amax
END
	Sub m1()
	Dim i As Integer
	Dim x As Integer
	Sheets("Лист1").Activate
	Dim q(5) As Integer
	For i = 1 To 5
	q(i) = Cells(i, 1).Value
	Next i
	x = q(1)
	For i = 1 To 5
	If q(i) > x Then x = q(i)
	Next i
	Cells(1, 2).Value = x
	End Sub
		a = [45, 17, 28, 34, 100, 25, 900, 18]
		i = 0
		max = a[i]
		while i < len(a):
		if a[i] > max:
		max = i
		max = a[i]
		i = i + 1
		print max

3.5. Сортировка массива методом «пузырька»

PascalABC:	Dev-C++ (потоковая библиотека):	Turbo Prolog:
Program zadacha5; Const m = 10; Var arr: array[1..m] of integer; i, j, k: integer;	#include <iostream> using namespace std; const MAX = 5; void Swap(int arr[], int pos1, int pos2)	domains list = integer* predicates bubble(list,list) up(list,list,integer)

```
 begin
   randomize;
write ('Исходный мас-
сив: ');
   for i := 1 to m do begin
    arr[i] := random(256);
      write (arr[i]:4);
   end;
   writeln; writeln;
    for i := 1 to m-1 do
      for j := 1 to m-i do
        if arr[j] > arr[j+1]
then begin
  k := arr[j];
  arr[j] := arr[j+1];
  arr[j+1] := k
        end;
    write      ('Отсортиро-
ванный массив: ');
   for i := 1 to m do
      write (arr[i]:4);
    writeln;
 readln
end.
```

```
{
  int tmp;
  tmp = arr[pos1];
  arr[pos1] = arr[pos2];
  arr[pos2] = tmp;
}
void PrintArray(int arr[])
{
 for(int i = 0; i < MAX;
++i)
    cout << arr[i] << endl;
}
 void          BubbleSort(int
arr[])
 {
  int i, j;
 for(i = 0; i < MAX - 1;
++i)
  for(j = 0; j < MAX - i;
++j)
      if (arr[j] > arr[j + 1])
        Swap(arr, j , j + 1);
 }
 int main()
 {
  int arr[MAX];
  int i;
  for  (i  =  0;  i  <  MAX;
++i)
    {
  cout << "Enter element:
" << endl;
      cin >> arr[i];
    }
  PrintArray(arr);
  BubbleSort(arr);
  cout  <<  "After  sort:  "
<< endl;
  PrintArray(arr);
  char r;
  cin >> r;
  return 0;
}
```

```
gen(integer,list)
goal
gen(20,L),
bubble(L,X),
write(X),exit.
clauses
bubble(L,X):-
up(L,L1,0),!,bubble(L1,
X).
bubble(L,L).
up([X,Y|L],[Y|L1],_):-
X>Y,!,up([X|L],L1,1).
up([X,Y|L],[X|L1],B):-
!,up([Y|L],L1,B).
up(L,L,1).
gen(1,[1]):-!.
gen(N,[N|L]):-M=N-
1,gen(M,L).
```

Small Basic:	VBA (Microsoft Excel):	Python:

```
dim mas(10)
 for i=0 to 9
input mas(i) ;
next i
print
for i=0 to 8
for j=0 to 8-i
if mas(j) > mas(j+1) then
a = mas(j)
mas(j) = mas(j+1)
mas(j+1) = a
        endif
        next j
next i
for i=0 to 9
print mas(i) + "  ";
next i
```

```
Option Explicit
 Sub test()
Dim MASS()
MASS = Application.
Transpose(Sheets(1)
.[A1].Resize(10, 1))
ArraySort MASS()
Sheets(1).[B1].Resize(10
, 1) = Applica-
tion.Transpose(MASS)
 End Sub
Sub Array-
Sort(ARRAY_())
Dim Exchange As Bool-
ean
Dim x As Integer
Dim t 'As Long
    Do
Exchange = False
For x =
LBound(ARRAY_) + 1
To UBound(ARRAY_)
 If ARRAY_(x - 1) >
ARRAY_(x) Then
 t = ARRAY_(x - 1)
       ARRAY_(x - 1) =
ARRAY_(x)
  ARRAY_(x) = t
  Exchange = True
         End If
       Next
   Loop While Exchange
End Sub
```

```
a=[1,0,9,5,3,8,4,6,2]
print (a)
n=len(a)
m=n-1
while m>0:
 for i in range(m):
  if (a[i]>a[i+1]):
   x=a[i]
   a[i]=a[i+1]
   a[i+1]=x
 m=m-1
print(a)
```