

## Содержание:

# ВВЕДЕНИЕ

Высокоуровневые языки появились сравнительно поздно, в 1970-х гг., т.е. примерно через 20 лет после появления компьютеров с современной архитектурой. Создание таких языков было обусловлено стремлением избавить программистов от трат времени и умственных усилий на рутинные операции, связанные с учетом особенностей тех или иных компьютерных архитектур [7].

Язык программирования (ЯП) высокого уровня - формализованная семантическая система, максимально приближенная к обычному человеческому языку или иным привычным знаковым системам (например, математическим формулам). Язык высокого уровня в минимальной степени привязан к процессору или операционной системе и направлен на то, чтобы программист сосредоточился на решении поставленной задачи, не отвлекаясь на особенности устройства компьютера [2].

Современные языки программирования принято относить к языкам программирования высокого уровня. Данный уровень говорит о том, что все эти языки максимально приближены к естественному человеческому языку. Этот факт позволяет облегчить процесс написания программ и предоставляет разработчикам целое множество возможностей.

Известно, что в основе всех языков программирования лежат простейшие конструкции - операторы, при помощи которых и создаются программы. Не смотря на многообразие языков программирования, основные алгоритмические конструкции присутствуют в каждом из них. К числу таких конструкций относятся:

- операторы присваивания;
- операторы ввода и вывода информации;
- операторы перехода;
- операторы выбора;
- операторы циклов.

Пользуясь этими конструкциями, программист может разработать программу любого уровня сложности.

Актуальность выбранной темы очевидна – ежедневно каждый из нас сталкивается с использованием компьютера и различными прикладными программами. Для того чтобы уметь самостоятельно разрабатывать программное обеспечение, требуется знать хотя бы один язык программирования. Однако большинство языков имеют схожий синтаксис в написании базовых конструкций. Именно поэтому важно изучить основные конструкции языков программирования высокого уровня, при помощи которых можно строить программы любой сложности.

Объектом исследования в данной работе являются языки программирования высокого уровня.

Предмет исследования – основные операторы языков программирования высокого уровня.

Цель работы – рассмотреть базовые конструкции языков программирования. Для достижения данной цели предстоит решить ряд задач:

- изучить историю развития языков программирования;
- привести классификацию языков программирования;
- дать определение основным понятиям языков программирования;
- описать основные операторы, используемые в языках программирования высокого уровня;
- привести примеры использования операторов.

При написании работы в качестве опорных источников были использованы следующие: М.А. Ревенко – «Практикум по программированию на языке TurboPascal» и Л.И. Долинер – «Основы программирования в среде PascalABC.NET».

язык программирование высокий оператор

# **ГЛАВА 1. ЯЗЫКИ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ**

## **1.1 Понятие и сущность высокого уровня**

В середине 50-х годов, когда вычислительная техника прочно укоренилась в университетах и научно-исследовательских лабораториях США и Европы,

наступило время стремительного прогресса в области программирования. Однако новые разработки отнюдь не отрицали всего того, что было сделано прежде. Напротив, они опирались на уже построенный фундамент. Компиляторы и интерпретаторы для так называемых языков ассемблера (такие языки требуют от программиста глубокого знания аппаратуры) остались важным средством программирования для любого компьютера. Однако, хотя эти средства и продолжали использоваться, их роль постепенно снижалась. Посредниками между программистами и машинами стали языки программирования нового типа. От языка ассемблера они отличались большей гибкостью и возможностью использования конструкций, подобных предложениям.

С появлением языков высокого уровня программисты получили возможность больше времени уделять решению конкретной проблемы, не отвлекаясь особенно на весьма тонкие вопросы организации самого процесса выполнения задания на машине. Кроме того, появление этих языков ознаменовало первый шаг на пути создания программ, которые вышли за пределы научно-исследовательских лабораторий и финансовых отделов.

Вначале 60-х годов все существующие языки программирования высокого уровня можно было пересчитать по пальцам. Однако вскоре их общее число (с учетом разнообразных диалектов) достигло нескольких сотен. Поэтому были предприняты различные попытки создать универсальный язык программирования. Ни одна из них не увенчалась полным успехом. Создается впечатление, что в программировании наилучший результат достигается только при индивидуальном подходе.

Среди десятка наиболее широко используемых языков каждый ориентирован на решение специфических задач.

Например, BASIC по-прежнему широко употребляется для написания простых программ, особенно программ для микрокомпьютеров.

FORTAN - с его четко определенными правилами выполнения арифметических действий - является классическим (чем-то вроде латыни или греческого) языком программирования, наиболее подходящим для выполнения естественнонаучных, математических и инженерных расчетов.

Язык программирования Кобол (COBOL, от COmmon Business Oriented Language - общий язык, ориентированный на деловые задачи), созданный в 1960 г. объединенным комитетом производителей и пользователей компьютеров, был задуман как основной язык для массовой обработки данных в сферах управления и бизнеса. В COBOL, в отличие от большинства других языков, все данные

описываются в отдельной секции, которая не совпадает с секцией команд. Это соглашение позволяет использовать совместно одни и те же описания данных в различных программах.

Другие языки еще более специализированны. Например, для обучения программированию школьников широко используется язык Logo (от греческого logos - слово). Этот язык позволяет новичку управлять движением по экрану специального символа под названием «черепашка». Так, набирая последовательно команды FORWARD 60 (вперед), LEFT 45 (влево), FORWARD 75, юный программист сообщает компьютеру, что он хочет нарисовать прямую линию длиной в 60 единиц, затем сделать поворот против часовой стрелки на 45 градусов и нарисовать еще одну линию длиной в 75 единиц.

Еще один заслуживающий внимания язык программирования - Алгол (ALGOL, от ALGOrithmic Language - алгоритмический язык). Как и COBOL, это - плод работы комитета, но не национального, а международного. Язык предназначен для записи алгоритмов, которые строятся в виде последовательности процедур, применяемых для решения поставленной задачи. Первая версия этого языка, ALGOL-58, была разработана в ходе напряженного восьмидневного совещания в конце весны 1958 г.

Специалисты по вычислительной технике и программированию из США (в их числе Джон Бекус - один из создателей языка FORTRAN) встретились со своими европейскими коллегами в Цюрихе. Цель встречи заключалась в разработке проекта универсального машинно-независимого языка, который по своему уровню не уступал бы FORTRAN. На самом деле FORTRAN и сам вполне мог бы претендовать на роль такого универсального языка, если бы не его чересчур тесная связь с фирмой IBM и ее машинами.

Программисты далеко неоднозначно приняли язык ALGOL-58 и его преемника ALGOL-60. Широкого одобрения (которого в конце концов добился FORTRAN) эти языки так и не получили. И все же их влияние на развитие других языков программирования оказалось весьма значительным.

Среди языков, при создании которых ставилась цель улучшить ALGOL, следует отметить Pascal. Этот язык был разработан в конце 60-х годов швейцарским ученым Никлаусом Виртом. Язык Pascal требует от программиста определения всех переменных в отдельной секции, расположенной в начале программы. Так как эти определения задаются явным образом, то в Pascal-программах сравнительно немного ошибок и их проще понять и исправить программисту, не являющемуся автором программы. Это делает Pascal весьма подходящим для создания больших

программ.

В 1963 г. он был объявлен официальным языком программирования для учащихся средних школ, которые намерены специализироваться в области вычислительной техники и программирования в американских университетах.

Несмотря на стремительный рост числа языков программирования, программное обеспечение существенно отставало в своем развитии от технической базы компьютеров. В то время как в области производства аппаратного обеспечения наблюдался неуклонный рост производительности и снижение цены, стоимость программного обеспечения продолжала увеличиваться. Иногда затраты на программное обеспечение больших новых вычислительных систем доходили до 80% от их общей стоимости. Это, естественно, вызывало недовольство инженеров, которые считали, что программное обеспечение фактически превратилось в фактор сдерживания как для развития аппаратного обеспечения компьютеров, так и для применения последних.

Главная причина, сделавшая программное обеспечение своего рода камнем преткновения, кроется в его все возрастающей сложности. Многие программы настолько велики, что даже специалисты не в состоянии их понять. Например, пакет программного обеспечения для управления системой воздушных перевозок, которая работает в 20 городах США и в одном городе Великобритании, насчитывает более 600 тыс. отдельных машинных команд. Из них 39 203 - команды условного перехода, обеспечивающие возможность разветвления программы по двум направлениям. Таким образом, общее количество возможных путей в программе составляет астрономическое число, которое приблизительно выражается числом 10 с 11 800 нулями.

(В языке BASIC оператор условного перехода может иметь вид `IF N >= 4 THEN 220`. Такой оператор предписывает компьютеру перейти к выполнению оператора, заданного в строке 220, если переменная N больше либо равна 4. Если N меньше 4, то компьютер должен автоматически перейти к выполнению следующей строки программы.)

Однако сложность программного обеспечения лишь частично обусловлена запутанностью решаемой задачи. Во многом она связана и с самой сутью процесса программирования, требующего огромного внимания и аккуратности при переводе конкретной задачи в термины, доступные компьютеру.

В программировании недопустимо полагаться на волю случая. Пользователь, замороженный магией компьютера, не владея искусством программирования, склонен упускать из виду, что те действия, которые он считает само собой разумеющимися, машине надо сообщать в самых мельчайших подробностях. Один специалист по компьютерам заметил: «Если говорить только о том, как человек обдумывает способ выполнения какого-либо действия, то здесь особых проблем не возникает. Однако люди пользуются такими удивительными способностями, как ассоциативное мышление и распознавание образов».

Для компьютера любое заранее планируемое действие надо разбить на самые элементарные шаги так, чтобы в результате получился алгоритм. В простейшем случае алгоритм ни чем не отличается от рецепта для приготовления праздничного пирога. Но программист должен определить и те элементарные шаги, которые человеческий разум склонен просто не замечать. Например, в кулинарном рецепте может быть сказано, что следует взять 3 яйца, однако там ничего не говорится о том, что они должны быть свежими и что их предварительно надо разбить и использовать только содержимое, без скорлупы.

Поскольку для того, чтобы предусмотреть любую случайность, надо написать сотни, а то и тысячи команд компьютера, естественно, растут затраты и вкрадываются ошибки. Все это порождает, по словам голландского ученого Эдсгера Дейкстры, «огромное множество досадных мелочей». Дейкстра - научный сотрудник фирмы «Бароуз» (BURROUGHS), один из наиболее авторитетных в мире теоретиков программирования и последовательный критик всего того, что он считает проявлением преступной халатности в данной области. Уже не один год он утверждает, что большинства ошибок можно избежать, и ведет неутомимую борьбу с небрежным стилем в программировании, которое развивалось в основном не как строгая наука, а как искусство, основанное на интуиции и личном опыте.

## **1.2 Развитие языков высокого уровня**

Высокоуровневые языки программирования, в плане создания ПО, стали всё по большей части удаляться от машинных кодов и реализовывать различные, помимо процедурного, парадигм программирования. К ним относят также и реализацию объектно-ориентированных принципов. C++, Java, Python, JavaScript, Ruby... - спектр языков данного типа наиболее популярен и востребован сегодня. Они предоставляют больше возможностей для реализации разнообразного ПО и нельзя

однозначно определить «специализацию» каждого из них. Но популярность применения в соответствующих областях обусловлена библиотеками/фреймворками для работы с ними, например:

JavaScript – Frontend. Язык был разработан для взаимодействия клиентского веб-браузера с пользователем и удалённым сервером. Наиболее популярные библиотеки: Angular, React и VUE. В данное время относительно активно употребляется и на web и т. п. серверах (backend), особенно популярен Node.js.

Ruby – Backend. Применяется для создания скриптов (служебных сервисных файлов) и на web серверах. Основной фреймворк - Ruby On Rails. Python – научная и инженерная сфера (помимо веб-области). Является альтернативой стандартным вычислительным и математическим пакетам (Mathematica, Octave, MatLab...), но имеет привычную семантику языка и большое число библиотек. Имеет много поклонников в области систем машинного обучения, статистики и искусственного интеллекта. Из часто используемых библиотек необходимо упомянуть django, numpy, pandas, tensorflow.

C++ – Универсал, эволюционное развитие языка C. Предоставляет возможности функционального и объектно-ориентированного программирования и не потеряв при этом способность низкоуровневого взаимодействия с аппаратным обеспечением. За счёт чего реализуется производительность и гибкость при создании ПО, но и цена соответствует: высокий порог вхождения за счёт сложной спецификации языка, необходимости самостоятельного контроля за ресурсами при выполнении программы. Многие однопользовательское и системное ПО написано с его применением: модули операционных систем (Windows, Symbian...), игры, редакторы (Adobe Photoshop, Autodesk Maya...), базы данных (MSSQL, Oracle...), проигрыватели (WinAmp...) и т. д.

Следует отметить, что современное ПО является сложным продуктом, в разработке которого используется сразу несколько языков программирования и расставлять степень участия каждого из них в общий результат бывает весьма затруднительно.

## **ГЛАВА 2. ОСНОВНЫЕ ОПЕРАТОРЫ**

Дальнейшее рассмотрение операторов языков программирования высокого уровня будем вести на примере языка Паскаль. Данный выбор обусловлен простотой языка и строгим синтаксисом.

## 2.1 Операторы присваивания

В языке Паскаль оператор присваивания обозначается двумя символами «:=», между которыми не ставится пробел. В левой части данного оператора должна стоять переменная, а в правой части – выражение, значение которого будет присвоено переменной.

Очень часто в Паскале можно увидеть конструкцию вида  $p:=p+1$ . Данная запись не содержит ошибки. Внутри компьютера данный оператор выполняется следующим образом: сначала берется исходное значение переменной, к которому прибавляется единица. После этих действий результат вычислений помещается в переменную  $p$ . Таким образом в языке Паскаль реализуется инкремент переменной.

Очень важно следить, чтобы все переменные, которые участвуют в правой части оператора присваивания, были определены к моменту его исполнения. В качестве правых частей для вычисления численных переменных операторы присваивания используют арифметические выражения, состоящие из переменных, констант, знаков операций, скобок и вызовов функций. В общем случае правила построения выражений аналогичны математической записи. Бинарные операции, применимые к целочисленным данным приведены в таблице 1 [19].

Для примера реализации операции присваивания рассмотрим задачу вычисления гипотенузы треугольника по двум известным катетам. Согласно теореме Пифагора, гипотенуза будет рассчитываться по формуле:

$$c = \sqrt{a^2 + b^2} \quad (1)$$

**Таблица 1 - Бинарные арифметические операции над типом integer**

Операция	Обозначение	Пример
Сложение	+	$p + 2$
Вычитание	-	$p - 2$



Умножение	*	$p * 2$
Деление нацело	div	$p \text{ div } 2$
Остаток от деления	mod	$p \text{ mod } 2$

Исходный код программы:

```
Program Op_prisv;
```

```
var
```

```
a,b,c:real;
```

```
begin
```

```
a:=3;
```

```
b:=4;
```

```
c:=sqrt(a*a+b*b);
```

```
end.
```

В данной программе использованы только операторы присваивания. При этом в двух случаях переменным просто присваивается начальное значение – это катеты треугольника. А в третьем случае происходит вычисление выражения, определяющего корень из суммы квадратов катетов.

В результате выполнения данного кода программа вычислит значение гипотенузы треугольника со сторонами  $a$ ,  $b$ , и занесет это значение в переменную  $c$ .

## 2.2 Операторы ввода-вывода

Ввод и вывод необходимы для связи программы с внешним миром – таким образом можно получать входные данные от пользователя и выводить полученные результаты на экран. Очевидно, программа без вывода не имеет смысла. В предыдущем примере было рассмотрено вычисление гипотенузы прямоугольного

треугольника, однако, без использования оператора вывода нельзя узнать результат, полученный при выполнении программы.

В языке Паскаль операторы ввода-вывода правильнее называть процедурами. Они служат для обмена данными между программой и внешними устройствами. Так, например, можно ввести данные с клавиатуры, из файла, вывести данные на экран или в файл.

Для ввода с клавиатуры в языке Паскаль существует два оператора: `Read` и `Readln`. Для вывода на экран – `Write` и `Writeln`. Дополнение «`ln`» произошло от английского слова «`line`» - строка, линия. Операторы, заканчивающиеся на «`ln`» в результате своих действий переводят курсор на новую строку. Так, например, при работе оператора `Write` курсор останется на следующей позиции после последнего выведенного символа. А в случае оператора `Read` очередные данные будут считываться из той же строки, где стоит курсор.

Традиционная запись данных операторов содержит параметры, однако, их может и не быть. В таком случае оператор `Writeln` будет реализовывать просто переход на новую строку, а оператор `Readln` будет ждать ввода любой клавиши.

Кроме стандартного вывода данных в языке Паскаль предусмотрен и форматированный вывод, который существует для того, чтобы сделать отображение на экране более понятным. Форматированный вывод содержит количество позиций, которые при выводе необходимо отвести под значение переменной [5].

В качестве примера использования операторов ввода-вывода модифицируем задачу определения гипотенузы прямоугольного треугольника следующим образом:

```
Program Op_vvod_vyvod;

uses crt;

var

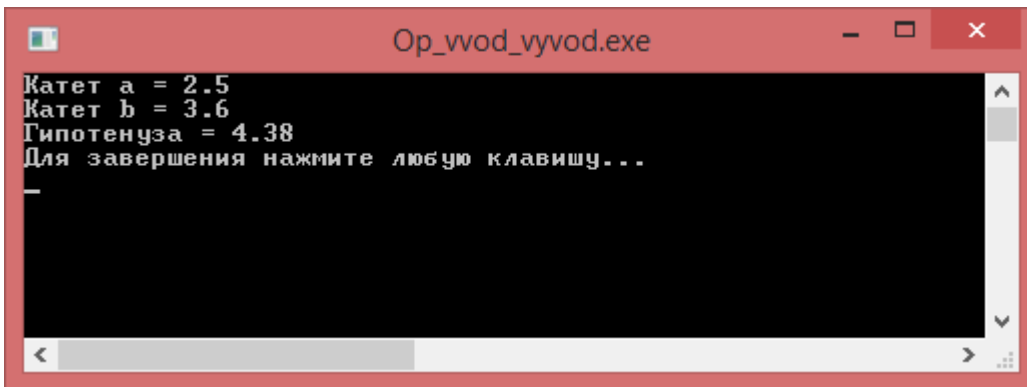
a,b,c:real;

begin

write('Катет a = ');
```

```
readln(a);  
  
write('Катет b = ');  
  
readln(b);  
  
c:=sqrt(a*a+b*b);  
  
writeln('Гипотенуза = ',c:3:2);  
  
writeln('Для завершения нажмите любую клавишу...');  
  
readln();  
  
end.
```

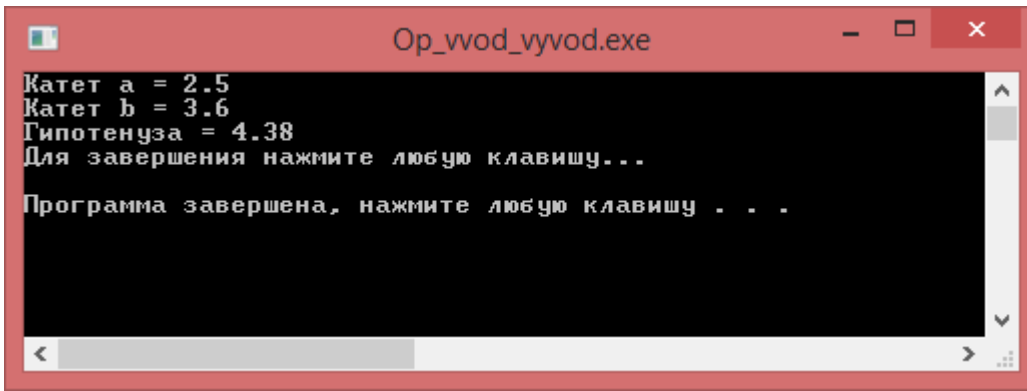
В данной программе используются операторы ввода исходных данных – катетов прямоугольного треугольника. Для вывода результата на экран используется форматированный вывод. Результаты работы программы приведены на рисунке 9.



```
Op_vvod_vyvod.exe  
Катет а = 2.5  
Катет б = 3.6  
Гипотенуза = 4.38  
Для завершения нажмите любую клавишу...  
_
```

**Рисунок 1 - Пример работы с операторами ввода-вывода**

Кроме того, в программе используется оператор Readln без параметров, который подтверждает завершение программы. Так, после нажатия любой клавиши программа выдаст сообщение о том, что ее работа завершена (см. рисунок 10).



```
Op_vvod_vyvod.exe
Катет a = 2.5
Катет b = 3.6
Гипотенуза = 4.38
Для завершения нажмите любую клавишу...
Программа завершена, нажмите любую клавишу . . .
```

**Рисунок 2 - Пример работы оператора ввода без параметров**

## 2.3 Операторы перехода

В языке Паскаль существует два вида операторов перехода – условный и безусловный.

Оператор безусловного перехода вызывает передачу управления оператору, которому предшествует соответствующая метка. Важно отметить, что данный подход не рекомендуется к использованию [16].

Рассмотрим пример программы с использованием оператора безусловного перехода:

```
Program Op_goto;
```

```
uses crt;
```

```
label m1;
```

```
var
```

```
a: integer;
```

```
begin
```

```
a:=5;
```

```
goto m1;
```

```
a:=a*10;
```

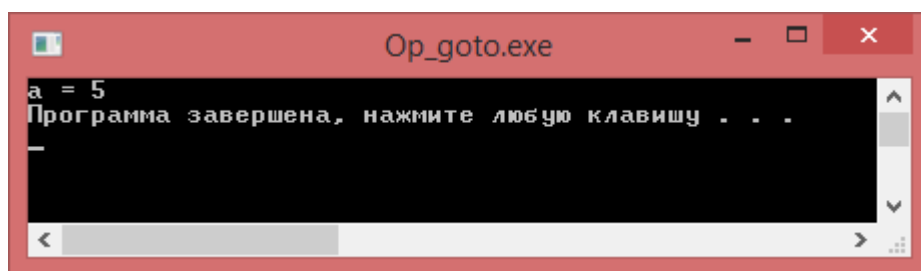
```
m1: writeln('a = ', a);
```

```
end.
```

Рассмотрим последовательность действий данной программы. В первую очередь переменной `a` присваивается значение 5. Затем в программе встречается оператор безусловного перехода, который посылает компьютеру сигналу о том, что следующий оператор, подлежащий исполнению, помечен меткой `m1`.

Следовательно, дальнейшие действия программы – вывод результата на экран, а не умножение значения переменной на 10. Важно отметить, что оператор `a:=a*10` в данной программе не исполнится никогда.

Результат исполнения данного кода приведен на рисунке 3.



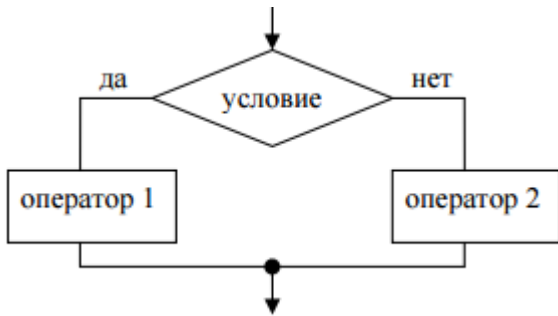
### **Рисунок 3 - Пример работы с оператором безусловного перехода**

Другой вид оператора перехода – условный переход. Данный оператор служит для выбора одной из двух альтернативных ветвей алгоритма в зависимости от значения некоторого условия [20].

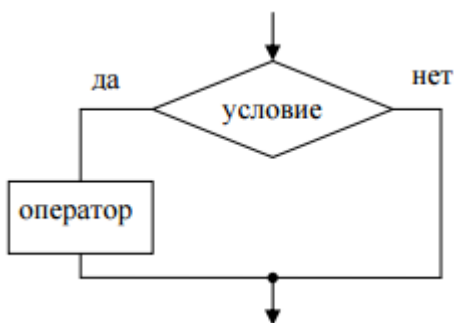
Условия в таких операторах представляют собой некоторые логические выражения, которые могут быть истинны, либо ложны. Выражения при этом могут быть простыми и сложными. Простое выражение включает в себя два операнда и операцию сравнения. Сложные представляют собой последовательность простых условий, которые объединены друг с другом знаками логических операций (больше, меньше, равно и т.п.). В языке Паскаль существует четыре логических операции:

- логическое сложение – `or`;
- логическое умножение – `and`;
- отрицание – `not`;
- исключающее «ИЛИ» – `xor`.

В языке Паскаль существуют условные операторы с одной и двумя ветвями. Если в условном операторе прописаны две ветви, такой оператор называется полным (см. рисунок 4), в противном случае – неполным (см. рисунок 4) [10].



**Рисунок 4 - Полное ветвление**



**Рисунок 5 - Неполное ветвление**

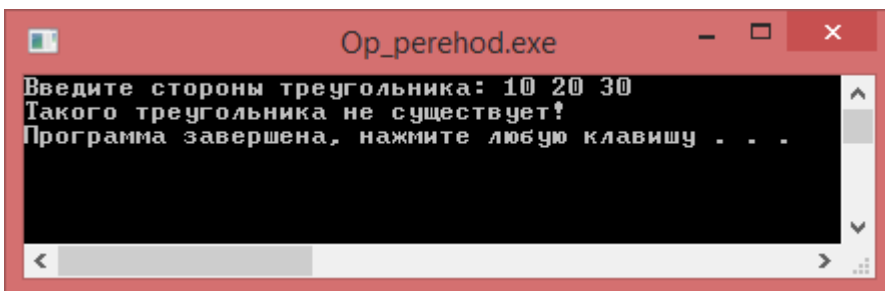
В качестве примера программы, использующей операторы ветвления, рассмотрим следующий код:

```
Program Op_perehod;  
  
uses crt;  
  
var  
a,b,c: integer;  
  
begin  
write('Введите стороны треугольника: ');  
read(a,b,c);  
if (a>=(b+c)) or (b>=(a+c)) or (c>=(a+b))
```

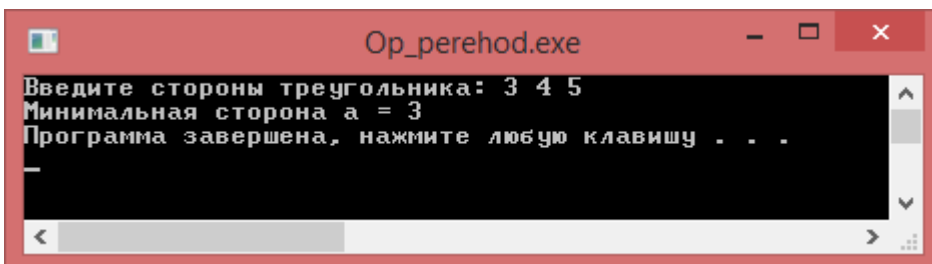
```
then writeln ('Такого треугольника не существует!')  
  
else  
  
begin  
  
if (a<b) and (a<c) then writeln ('Минимальнаясторонаа = ',a)  
else if (b<a) and (b<c) then writeln ('Минимальнаясторона b = ',b)  
else writeln ('Минимальная сторона c = ',c);  
  
if ((a+b+c)>50) then writeln('Периметрбольше 50');  
  
end;  
  
end.
```

В данной программе реализован запрос ввода данных сторон треугольника. Далее, происходит проверка введенных данных – существует ли треугольник с такими сторонами. В том случае, если он не существует, пользователь получает соответствующее сообщение (см. рисунок 15), иначе – программа определяет минимальную сторону и выводит ее на экран.

Для примера использования неполного ветвления реализована проверка периметра треугольника. Так, если он больше 50, программа выдаст соответствующее сообщение, в противном случае просто завершит свою работу.



**Рисунок 6 - Пример работы оператора полного ветвления**



## Рисунок 7 - Пример работы оператора неполного ветвления

### 2.4 Операторы выбора

В том случае, когда в условном операторе необходимо использовать больше, чем две ветви, можно пойти двумя путями:

- использовать вложенные условия;
- использовать оператор выбора.

Очевидно, использование вложенных условий является не самым удобным способом решения подобных задач, поэтому в языке Паскаль был создан оператор выбора Case, синтаксис которого выглядит следующим образом:

```
Case<ключ>of  
  
C_1: <операторы_1>;  
  
C_2: <операторы_2>;  
  
...  
  
C_N: <операторы_N>;  
  
else<операторы 0>  
  
end;
```

В данной записи ключом называется выражение порядкового типа, в зависимости от значения которого и принимается решение. Значениями C\_1, C\_2, ..., C\_N обозначены константы - возможные варианты значения ключа, которые необходимо рассмотреть при вычислениях. Данным значениям соответствуют блоки операторов, которые исполняются, если значение ключа совпадает со значением C. Если же значение ключа не совпало ни с одной из указанных констант, выполнится блок <операторы\_0>. Важно отметить, что этот блок может отсутствовать [15].

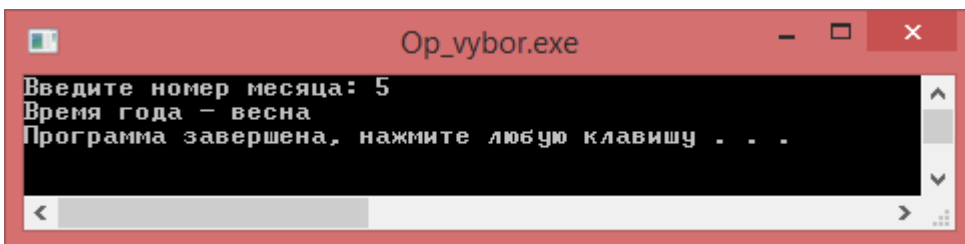
В качестве примера использования оператора выбора рассмотрим задачу определения времени года по введенному месяцу[13]:

```
Program Op_vybor;
```



```
uses crt;  
  
var  
  
m: integer;  
  
begin  
  
write('Введите номер месяца: ');  
  
readln(m);  
  
case m of  
  
1,2,12: writeln('Время года - зима');  
  
3,4,5: writeln('Время года - весна');  
  
6,7,8: writeln('Время года - лето');  
  
9,10,11: writeln('Время года - осень');  
  
else writeln('Такого месяца не существует!');  
  
end;  
  
end.
```

В данной программе реализован оператор выбора, который в качестве констант использует сразу несколько значений, операторы для которых идентичны. Данный подход является очень удобным и позволяет сократить число ветвей программы с 13 до 5. Результат выполнения приведенного кода представлен на рисунке 8.



**Рисунок 8 - Пример работы с оператором выбора**

## **2.5 Операторы цикла**

Циклическим называется такой алгоритм, в котором задана некоторая последовательность действий для многократного исполнения. Эта последовательность действий называется телом цикла. Заметим, что тело цикла указано в алгоритме один раз, но исполняться оно может многократно. Однократное исполнение тела цикла называется итерацией. В языках программирования высокого уровня, как правило, используют три вида циклов:

- цикл с предусловием – while. Состоит из условия цикла и его тела (см. рисунок 9). Пока условие цикла будет иметь значение «истина», тело цикла будет исполняться.



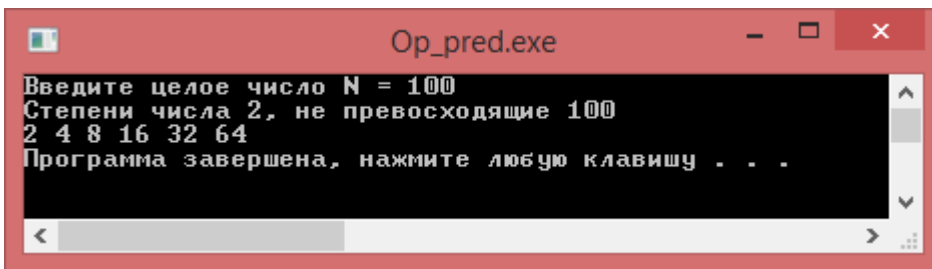
**Рисунок 9 - Блок-схема цикла с предусловием**

Как видно из блок-схемы, если условие цикла с самого начала имеет значение «Ложь», то тело цикла ни разу не будет исполнено. Если в процессе исполнения цикла условие всегда принимает значение «Истина», то цикл начинает исполняться бесконечно – происходит заикливание. Это означает, что в алгоритме допущена ошибка. Пример использования цикла с предусловием:

```
Program Op_pred;  
  
uses crt;  
  
var  
  
N, p: integer;  
  
begin  
  
write('Введите целое число N = ');  
  
readln(N);
```

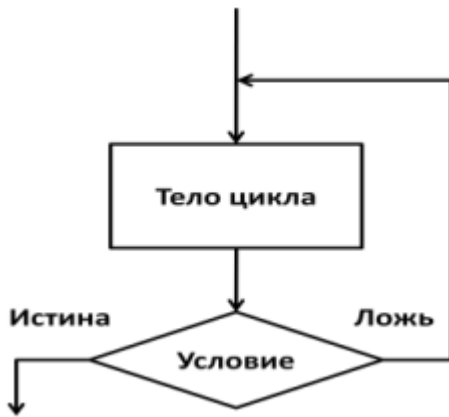
```
writeln('Степени числа 2, не превосходящие ', N);  
  
p:= 2;  
  
while p <= N do  
  
begin  
  
write(p, ' ');  
  
p:= p * 2;  
  
end;  
  
writeln();  
  
end.
```

Данная программа запрашивает у пользователя целое число, а затем выводит все степени двойки, которые не превосходят заданное число. Результат работы программы представлен на рисунке 10 [1];



**Рисунок 10 - Пример работы цикла с предусловием**

- цикл с постусловием – repeat-until. В данном случае сначала выполняется тело цикла, после чего проверяется выражение, записанное в блоке until (см. рисунок 11).



**Рисунок 11- Блок-схема цикла с постусловием**

В том случае, когда условие цикла выполнено, его тело больше не повторяется, и программа переходит к следующему оператору. Важно отметить, что тело данного цикла выполняется как минимум один раз. Примеры использования данного цикла:

```

ProgramOp_post;

uses crt;

var m, N, sum: integer;

begin
write('N = ');

read(N);

m:=0;

sum:=0;

repeat

m:=m+1;

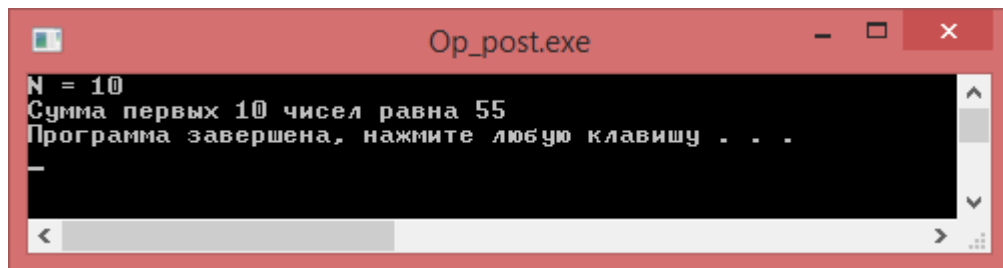
sum:=sum+m;

until m=N;

writeln ('Сумма первых ',N,' чисел равна ', sum);

end.
  
```

В данной программе определяется сумма первых N натуральных чисел. N задается пользователем с клавиатуры [12];



**Рисунок 12 - Пример работы цикла с постусловием**

- цикл с параметром – for. Данный цикл выполняется заданное количество раз. Существует две формы записи данного цикла в зависимости от того, каким образом должна изменяться переменная цикла. Если она должна увеличиваться, используется запись:

```
for<переменная_цикла>:=  
<начальное_значение>to<конечное_значение>do<операторы>.
```

Если же переменная цикла должна уменьшаться, используется другая запись:

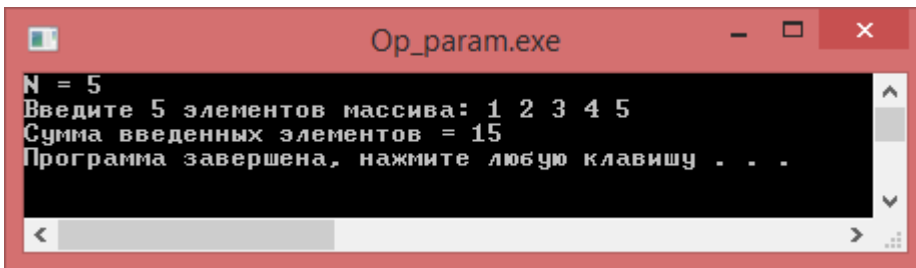
```
for<переменная_цикла>:=  
<начальное_значение>downto<конечное_значение>do<операторы> [14].
```

Чаще всего данный цикл используется при обработке массивов. Рассмотрим пример подобной программы:

```
Program Op_param;  
  
uses crt;  
  
var i, n, sum, a: integer;  
  
begin  
  
write('N = ');  
  
readln(n);  
  
write('Введите ',n,' элементов массива: ');  
  
sum:=0;
```

```
for i:=1 to n do  
  
begin  
  
read(a);  
  
sum:=sum+a;  
  
end;  
  
writeln('Сумма введенных элементов = ', sum);  
  
end.
```

В данной программе у пользователя запрашивается размерность массива, а затем и сам массив. В результате выполнения программы считается сумма введенных элементов. Результат выполнения приведенного кода представлен на рисунке 13.



**Рисунок 13 - Пример работы цикла с параметром**

В данной главе рассмотрены основные операторы языков программирования высокого уровня на примере языка Паскаль.

## **ЗАКЛЮЧЕНИЕ**

Языки программирования высокого уровня - языки, которые используют различные абстрактные смысловые конструкции, которые просто невозможно сообщать машине на низкоуровневых языках в силу их большого объема и сложности.

Программирование незаметно стало одной из важнейших сфер деятельности нашего времени. Все люди активно пользуются компьютерной техникой, смартфонами, интернетом, самыми разнообразными гаджетами и все это просто не может существовать без специально написанных программ. Именно от качества используемого ПО зависит конечное быстродействие, стабильность работы,

функциональность и многие другие параметры практически любого оборудования.

Первые языки программирования появились достаточно давно, еще примерно в середине 20-го века. Конечно, они были достаточно примитивны по современным меркам, но вполне справлялись с поставленными на них задачами. Сейчас существуют разные виды языков программирования, к примеру, языки низкого и высокого уровня. Каждый из них необходим для решения определенного спектра задач. Помимо указанных вариантов, есть еще алгоритмические, формальные, машинные, символические, императивные и многие другие типы языков программирования, но наибольшее распространение и актуальность сейчас имеют именно языки низкого и высокого уровня. Даже указанной информации достаточно, чтобы понять: классификация языков программирования – это очень объемное и сложное занятие, которое может растянуться на многие часы.

Языки программирования низкого уровня обращаются непосредственно к «железу», давая ему определенные точные команды, а языки программирования высокого уровня оперируют более абстрактными понятиями, здесь не нужно задавать способ работы каждой детали устройства, а достаточно просто в общих чертах задавать выполнение определенных задач и функций. Программисты, работающие с языками низкого уровня, должны знать в придачу ко всему еще и основы электроники, технические нюансы устройств, с которыми они работают, для работы на высоком языке все это не так нужно.

К примеру, если подключенное к ПК устройство (например, кулер) может работать на максимальных оборотах 1000 об/мин, то когда программист, используя низкоуровневый язык, пишет на него драйвера, он должен это знать и учитывать, так как если он поставит большее количество оборотов, то устройство испортится. Программисты же, которые используют языки программирования высокого уровня даже не задумываются об этом – они просто задают, к примеру, запуск кулера в определенный момент времени и его остановку. Но стоит помнить, что это довольно упрощенные определения.

Если говорить об отличиях с другими видами языков, то они могут быть еще более существенны. К примеру, разница между машинным кодом и высокоуровневым языком, как между небом и землей – в первом случае нужно работать с понятным машине шифром и набором принятых обозначений, во втором – использовать абстрактный язык с собственными правилами и синтаксисом.

Ответить конкретно на вопрос о том, когда именно появился первый язык программирования высокого уровня довольно сложно. Первые попытки внедрить что-то подобное наблюдались еще в 70 годах, но тогда массового использования в основном Pascal, который еще нельзя отнести к высокому уровню. Американские военные первыми взялись разрабатывать язык программирования высокого уровня для решения своих задач. В результате их работы в начале 80-х годов ими был разработан язык Ada, который был очень функциональным для своего времени, но в то же время предельно упрощен. Его главной задачей было программирование различной военной аппаратуры, встроенных систем, где любые сложности и долгие подсчеты идут только во вред.

Также приблизительно в те же годы был создан всеми любимый язык C, с которого в итоге развились и C++ и Си Шарп, и ряд других достойных примеров, список которых может оказаться довольно длинным. Также именно из высокоуровневого языка C берет свое начало популярнейший в наше время язык высокого уровня Java, на котором одинаково эффективно пишутся программы, скрипты, плагины и прочие «примочки» как на компьютеры, так и на разнообразные гаджеты: смартфоны, планшеты, смарт часы, очки виртуальной и дополненной реальности. Указанные языки были лидерами еще в далекие 80-90-е годы и остаются ими и поныне, хотя конечно, за это время изменилось и появилось очень многое.

## **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Анисимов А.Е. Практикум по основам программирования. – Ижевск: Изд-во «УУДмуртский университет», 2014. – 95 с.
2. Баженова И.Ю. Языки программирования. – М.: Издательский центр «Академия», 2012. – 368 с.
3. Бузыкова Ю.С. Языки и технологии программирования / Ю.С. Бузыкова, Т.А. Жданова, М.А. Шувалова. – Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2014. – 44 с.
4. Волкова И.А. Основы объектно-ориентированного программирования. Язык программирования C++ / И.А. Волкова, А.В. Иванов, Л.Е. Карпов. – М.: Издательский отдел факультета ВМК МГУ, 2011. – 112 с.
5. Грацианова Т.Ю. Программирование в примерах и задачах. – М.: БИНОМ, 2015. – 354 с.
6. Диканев Т.В. Принципы и алгоритмы прикладного программирования / Т.В. Диканев, С.Б. Вениг, И.В. Сысоев. – Саратов: Изд-во Саратов. ун-та, 2012. – 140 с.



7. Долинер Л.И. Основы программирования в среде PascalABC.NET. – Екатеринбург: Изд-во Урал. ун-та, 2014. – 128 с.
8. Дронова Е.Н. Основные алгоритмические модели: учебное пособие. – Барнаул: АлтГПУ, 2016. – 158 с.
9. Зюзьков В.М. Программирование. – Томск: Эль Контент, 2013. – 186 с.
10. Котликова В.Я. Введение в Турбо Паскаль. – Курган: Изд-во Курганского гос. ун-та, 2016. – 32 с.
11. Лучников В.А. Программирование на языке Паскаль. – Иркутск: ИргУПС, 2014. – 168 с.
12. Овчинников А.А. Основы программирования на Паскаль ABC. – Волгоград: Изд-во МОУ СОШ № 95, 2012. – 27 с.
13. Ожикенов К.А. Сборник задач для проведения лабораторно-практических занятий по дисциплине «Программирование». – Алматы: Изд-во УИЯиДК, 2011. – 135 с.
14. Попов Е.А. Экспресс курс программирования в Lazarus. – СПб.: Университет ИТМО, 2015. – 76 с.
15. Ревенко М.А. Практикум по программированию на языке TurboPascal. – Воронеж: ВГПУ, 2012. – 65 с.
16. Стенгач М.С. Лекционный курс электронного курса «Информатика». – Самара: Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С.П. Королева. – 35 с.
17. Тюгашев А.А. Основы программирования. – СПб.: Университет ИТМО, 2016. – 160 с.
18. Федоров Д.Ю. Основы программирования на примере языка Python. – СПб.: Питер, 2016. – 176 с.
19. Цветков А.С. Язык программирования Pascal. Система программирования ABCPascal. – Санкт-Петербург: Павловск, Изд-во Царского лицея, 2016. – 46 с.
20. Ширяева Е.В. Практикум по курсу «Основы информатики» / Е.В. Ширяева, М.Н. Романов, Т.Ф. Долгих. – Ростов-на-Дону: Изд-во ЮФУ, 2015. – 233 с.