# Министерство Образования Республики Беларусь

Учреждение Образования «Гомельский государственный университет им. Ф. Скорины»

## Математический факультет

### Кафедра ВМ и Программирования

# Контрольная работа

**Тема:**

**«Способы описания алгоритма. Виды операторов»**

**1. Алгоритм и его свойства. Способы описания алгоритма**

Для пояснения понятия «алгоритм» важное значение имеет определение понятия «исполнитель алгоритма». Алгоритм формулируется в расчёте на конкретного исполнителя; алгоритм является руководством к действию для исполнителя, поэтому значение слова «алгоритм» близко по смыслу к значению слов «указание» или «предписание». Можно сказать, что алгоритм – понятное и точное предписание исполнителю совершить определённую последовательность действий для достижения указанной цели или решения поставленной задачи или алгоритм – точное предписание, которое задаёт вычислительный процесс, начинающийся с произвольного исходного данного из некоторой совокупности возможных для этого процесса данных и направленный на получение полностью определяемого этими исходными данными результата.

Основные свойства алгоритма.

1. Алгоритм имеет некоторое число входных величин – аргументов, задаваемых до начала исполнения. Цель выполнения алгоритма – получение результата, имеющего вполне определённое отношение к исходным данным. Для алгоритма можно выбирать различные наборы входных данных из множества допустимых для этого процесса данных, т. е. можно применять алгоритм для решения целого класса задач одного типа, различающихся исходными данными. Это свойство алгоритма называют *массовостью*. Однако существуют алгоритмы, применимые только к единственному набору данных. Тогда свойство массовости означает применимость алгоритма ко всем объектам этого класса.
2. Чтобы алгоритм можно было выполнить, он должен быть понятен исполнителю. *Понятность* *алгоритма* означает знание исполнителя о том, что надо делать для исполнения этого алгоритма.
3. Алгоритм представляется *в виде конечной последовательности* шагов, (алгоритм имеет дискретную структуру) и его исполнение расчленяется на выполнение отдельных шагов.
4. Каждый шаг алгоритма должен быть чётко и недвусмысленно определён и не должен допускать произвольной трактовки исполнителем. Алгоритм рассчитан на чисто механическое исполнение. Именно *определённость* *алгоритма* даёт возможность поручить его исполнение автомату.
5. Выполнение алгоритма заканчивается после выполнения *конечного числа шагов*. При выполнении алгоритма некоторые его шаги могут *повторяться* многократно.
6. Каждый шаг алгоритма должен быть выполнен точно и за конечное время. Алгоритм должен быть *эффективным*.
7. **Линейные и ветвящиеся вычислительные процессы**
8. Линейный – это такой вычислительный процесс, в котором самостоятельные этапы вычисления выполняются в линейной последовательности.
9. Ветвящийся – это процесс, реализация которого в зависимости от исходных данных или промежуточных результатов происходит по одному из нескольких, заранее определяемых направлений, выбор той или иной ветви вычислений осуществляется проверкой логического условия, определяющего свойства исходных данных или промежуточных результатов.

**3. Основные понятия языка Паскаль**

Программа на языке Паскаль формируется с помощью конечного набора знаков, образующих *алфавит* языка, и состоит из букв, цифр, специальных символов.

В качестве *букв* используются прописные и строчные буквы латинского алфавита и знак подчёркивания; в качестве *цифр:* арабские цифры от 0 до 9.

При написании программ применяются *специальные символы*: +, -, \*, /, =, <, >, [], (), @, {},:,;’, # (номер), $ (знак денежной единицы), ^ (тильда), пробел, точка и запятая.

Неделимые последовательности знаков алфавита образуют слова, отделённые друг от друга разделителями и несущими определённый смысл в программе. Разделителем может служить пробел, символ конца строки, комментарий. Слова подразделяются на зарезервированные, стандандартные идентификаторы и идентификаторы пользователя.

*Зарезервированные слова* являются составной частью языка и их нельзя использовать в качестве идентификаторов. В языке Паскаль зарезервированными являются следующие слова: and, array, begin, case, const, div, do, downto, else, end, file, for, forward, function, goto, if, in, lable, mod, nil, not, of, or, packed, procedure, program, record, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor.

*Стандартные идентификаторы* служат для обозначения заранее определённых разработчиками языка типов данных, констант, процедур и функций.

*Идентификаторы пользователя* используются для обозначения меток, констант, типов, переменных, процедур и функций, определённых самим программистом.

**4. Общая структура программы. Описание меток, определение констант, определение типов, описание переменных**

Структура программы.

**Program** <имя программы>;

{Раздел описаний}

**Uses** {подключаемые модули}

**Label** {объявление глобальных меток}

**Const** {объявление констант}

**Type** {объявление типов}

**Var** {объявление переменных}

**Procedure** {описание процедур}

**Function** {описание функций}

{Раздел операторов}

**Begin**

{операторы}

**End**.

Все данные, в зависимости от способа их хранения и обработки можно разделить на две группы константы и переменные.

*Константами* называются элементы данных, значения которых установлены в описательной части программы и в процессе выполнения программы не изменяются.

Стандартные виды констант:

1. *Целочисленные* – определяются посредством чисел, записанных в десятичном или шестнадцатеричном формате, не содержащих десятичной точки.

2. *Вещественные* – определяются посредством чисел, записанных в десятичном формате данных.

3*. Символьные* – это любой символ персонального компьютера, заключённый в апострофы.

1. *Строковые* – определяются последовательностью произвольных символов, заключённых в апострофы.
2. *Типизированные* – переменные с начальным значением. Каждой типизированной константе ставится в соответствие имя, тип и начальное значение.
3. *Зарезервированные константы*.

Формат описания констант:

**Const**

Идентификатор=значение;

Типы данных.

*Тип* – это множество значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими значениями.

Переменные в отличие от констант могут менять свои значения в процессе выполнения программы. Тип констант автоматически распознаётся компилятором без предварительного описания. Тип переменной должен быть описан перед тем, как с переменными будут выполняться какие-либо действия.

Формат описания переменных:

**Var**

Идентификатор: тип;

1. **Скалярные типы данных: стандартные и описанные пользователем**

*Логический тип*. Значениями логического типа может быть одна из констант False или True.

*Целые типы*. Диапазон возможных значений целых типов зависит от их внутреннего представления.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Название | Длина, байт | Диапазон значений |
| Byte | Длиной в байт | 1 | 0. 255 |
| ShortInt | Короткое целое | 1 | -128..127 |
| Word | Длиной в слово | 2 | 0..65535 |
| Integer | Целое | 2 | -32768..32767 |
| LongInt | Длинное целое | 4 | -2147483648..2147483647 |

*Символьный тип*. Значениями символьного типа является множество всех символов ПК. Для кодировки используется код ASCII (American Standard Code for Information Interchange).

*Перечисляемый тип*. Задаётся перечислением тех значений, которые он может получить. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамлённом круглыми скобками.

Переменные перечисляемого типа можно объявлять без предварительного описания типа.

*Тип-диапазон*. Тип-диапазон есть подмножество своего базового типа, в качестве которого может выступать любой скалярный тип, кроме вещественного и типа-диапазона. Тип-диапазон задаётся границами своих значений внутри базового типа.

Тип-диапазон можно непосредственно указывать при объявлении переменной.

*Вещественные типы*. Значения вещественных типов определяют произвольное вещественное число с некоторой конечной точностью, зависящей от внутреннего формата числа.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип | Название | Длина, байт | Кол-во цифр мантиссы | Диапазон десятичного порядка |
| Real | Вещественный | 6 | 11..12 | -39..38 |
| Single | С одинарной точностью | 4 | 7..8 | -45..38 |
| Double | С двоичной точностью | 8 | 15..16 | -324..308 |
| Extended | С повышенной точностью | 10 | 19..20 | -4932..4932 |
| Comp | Сложный | 8 | 10..20 | -2\*10 +1-2\*10 -1 |

1. **Простые операторы: присваивания, перехода Goto, пустой оператор. Простейший ввод-вывод**

Операторы выполняются в том порядке, в котором они записаны в программе. Разделителем оператора служит точка с запятой.

Все операторы разделяются на две группы: простые и структурные.

Операторы, не содержащие внутри себя других операторов, называются *простыми*. К ним относятся операторы присваивания, безусловного перехода, пустой оператор и оператор вызова процедур.

Оператор присваивания выполняет выражение, заданное в его правой части, и присваивает результат переменной, идентификатор которой расположен в левой части.

Формат оператора:

Идентификатор:=выражение;

*Оператор безусловного перехода Goto* служит для передачи управления оператору, помеченному меткой. Метка отделяется от оператора двоеточием. Оператор Goto применяется в случае, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-либо другой, отмеченный меткой оператор.

Формат оператора:

**Goto** метка;

Формат описания меток:

**Label**

Имя метки;

*Пустой оператор* не содержит ни одного символа и не выполняет никаких действий.

Для выполнения операций ввода-вывода служат 4 процедуры: Read, Readln, Write, Writeln.

*Процедура чтения Read* обеспечивает ввод числовых данных, символов, строк и т. д. для последующей их обработки программой.

Формат:

Read (x1, x2,…, xn);

где x1, x2, …, xn – переменные допустимых типов.

*Процедура чтения Readln* аналогична процедуре Read. Единственное отличие заключается в том, что после считывания последнего в списке значения для одной процедуры Readln данные для следующей процедуры Readln будут считываться с начала новой строки.

*Процедура записи Write* производит вывод числовых данных, символов, строк и булевских значений.

Формат:

Write (y1, y2,…, yn);

где y1, y2, …, yn – выражения целочисленного, вещественного, символьного, строкового, булевского и др. типов.

*Процедура записи Writeln* аналогична процедуре Write, но после вывода последнего в списке значения для текущей процедуры Writeln происходит перевод курсора к началу следующей строки. Процедура Writeln, записанная без параметров, вызывает перевод строки.

1. **Структурные операторы: условный оператор If, составной оператор Begin-End, оператор выбора Case**

*Структурные операторы* представляют собой структуры, построенные из других операторов по строго определённым правилам.

*Составной оператор* представляет собой группу из произвольного числа операторов, отделённых друг от друга точкой с запятой и ограниченную операторными скобками Begin и End.

Формат оператора:

#### Begin

Оператор 1;

…

оператор N;

**End**;

*Условные операторы* обеспечивают выполнение или невыполнение некоторого оператора, группы операторов или блока в зависимости от заданных условий.

*Оператор условия If* может принимать одну из форм:

1. **If** условие **then** оператор1 {полная условная конструкция}

**Else** оператор2;

1. **If** условие **then** оператор; {неполная условная конструкция}

*Оператор выбора Case* является обобщением оператора If и позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого селектором, и списка параметров, каждому из которых предшествует список констант выбора. Как и в операторе If, здесь может присутствовать слово Else, имеющее тот же смысл.

Формат оператора:

**Case** выражение-селектор **of**

Список 1: оператор 1;

…

Список N: оператор N

**Else** оператор

**End**