Основы алгоритмизации

Краткое содержание

Курс строится от планирования решения и разработки алгоритма к программе на C++, её отладке и эффективной реализации.

Организация курса

Курс рассчитан на **два семестра** и включает лекции и лабораторные работы. В первом семестре — **17 лекций** и **14 лабораторных занятий**. Первая методичка содержит **7 лабораторных работ**, поэтому на каждую тему отводится по **2 занятия**.

Оценки формируются по лабораторным работам: дополнительных письменных или устных контрольных не предусмотрено. Задание и свой вариант можно получить заранее и сдать работы досрочно, но откладывать выполнение до конца семестра рискованно.

На лабораторные нужны:

- методичка в бумажном или электронном виде;
- тетрадь с разобранным материалом;
- флешка или другой носитель для хранения работ.

Отчёт оформляется по требованиям преподавателя. В типичном отчёте нужны **описание алгоритма**, **блок-схема** и **текст программы**. Дополнительные источники использовать можно , но справочную информацию следует проверять: автоматические помощники не являются абсолютным источником знаний.

В первом семестре изучаются инструменты языка: ввод и вывод данных, условия, циклы, массивы, файлы и другие базовые конструкции. Сложные алгоритмы переносятся на второй семестр, где базовые инструменты используются для алгоритмизации. Тема комбинированных структур данных также частично переносится на второй семестр.

Этапы решения задачи

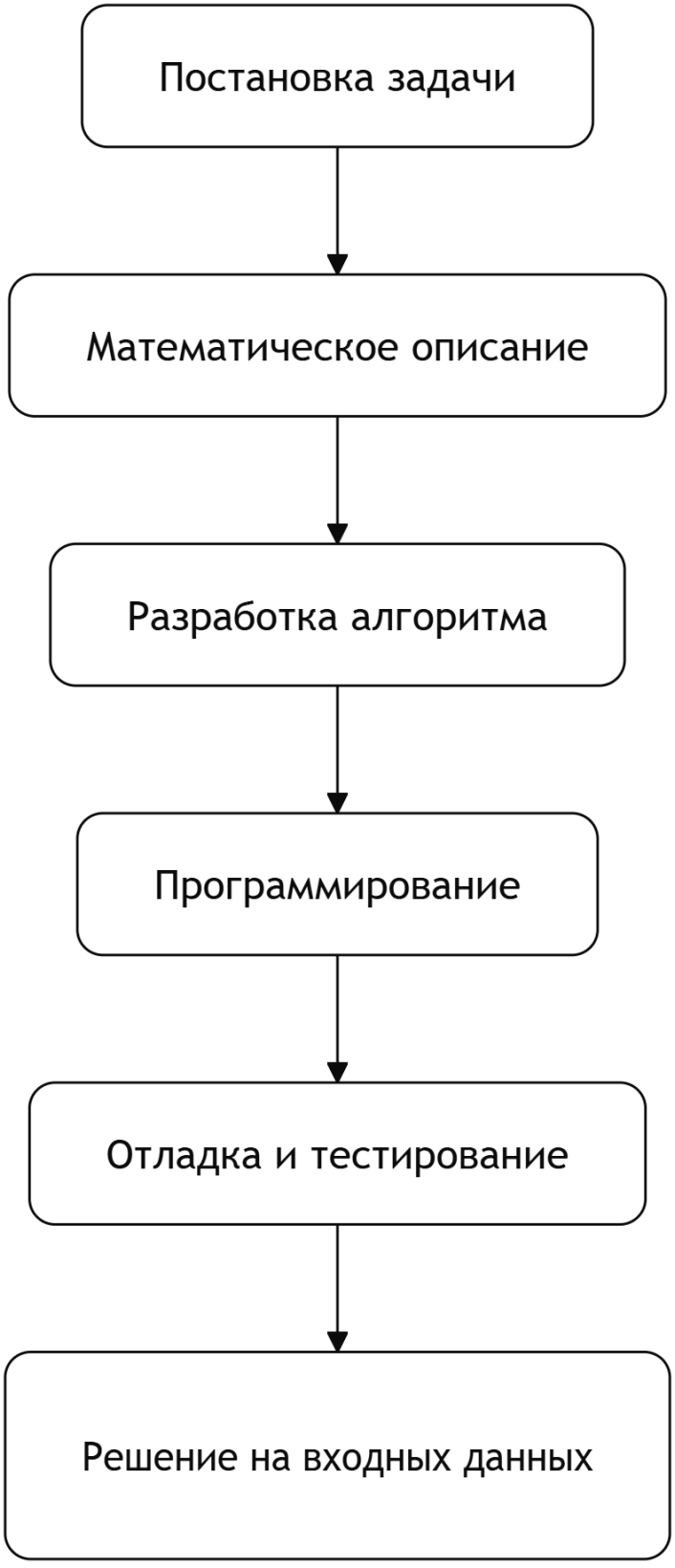
Нельзя начинать решение с написания кода. Сначала нужно спроектировать решение, затем перенести его на язык программирования.

1. **Постановка задачи** — чётко определить, что дано, что требуется найти и какие ограничения действуют. Например, для квадратного уравнения должны быть заданы коэффициенты и требуемый результат.
2. **Математическое описание** — выбрать формулы и метод решения. Если способов несколько, предпочтителен более короткий и оптимальный.
3. **Разработка алгоритма** — разложить решение на отдельные действия и определить порядок их выполнения.
4. **Разработка программы** — записать алгоритм на языке программирования.

Сделано на сайте Зачёт | zachet.app

5. **Отладка** — найти и устранить ошибки.
6. **Решение задачи** — запустить проверенную программу на произвольных допустимых исходных данных.

Одна успешная проверка не доказывает правильность программы. Нужно заранее подготовить разные тесты: обычные, граничные и ошибочные случаи. Если решение невозможно, программа должна сообщить об этом, а не зацикливаться или зависать.



Сделано на сайте Зачёт | zachet.app

Важно оценивать **эффективность**: время работы, объём памяти и сложность кода. Программа может выдавать правильный ответ, но использовать неоправданно много операций или ресурсов . Даже в учебных задачах следует искать не первое работающее, а аккуратное и эффективное решение.

Понятие алгоритма

> ***Алгоритм*** *— чёткое описание последовательности действий, которые необходимо выполнить для решения задачи.*

Алгоритмы встречаются не только в программировании: инструкции по установке приложения и действия при снятии денег в банкомате также задают последовательность шагов.

Свойства алгоритма

Свойство	Смысл
Дискретность	Решение разбито на отдельные простые шаги.
Определённость	Каждое действие сформулировано однозначно и не допускает разных толкований.
Массовость	Алгоритм применим к целому классу задач с разными исходными данными.
Результативность	Алгоритм за конечное число шагов выдаёт решение или сообщение о невозможности его получить.
Эффективность	Используются разумные объёмы времени, памяти и других ресурсов.

Бесконечный цикл, зависание или отсутствие сообщения об ошибке нарушают результативность . Эффективность особенно важна в реальных программах, где объём данных и цена каждой операции могут быть большими.

Способы записи алгоритмов

Алгоритм записывают тремя основными способами:

1. **Естественный язык** — словесное описание решения без команд и синтаксиса конкретного языка. Это первый способ, с которого удобно начинать: нужно понять, какие данные взять, что сравнить, сложить или умножить.
2. **Блок-схема** — графическое представление последовательности действий. Она не зависит от языка программирования и показывает логику решения в виде универсальной инструкции.
3. **Алгоритмический язык** — текст программы. Он обязательно включается в отчёт и реализует построенный алгоритм.

Сделано на сайте Зачёт | zachet.app

Блок-схема полезна, пока алгоритм трудно удерживать в голове: она показывает, какой шаг следует за другим, где происходит проверка и где начинаются повторения.

Типовые структуры алгоритмов

> *Типовые структуры — ограниченный набор блоков и стандартных способов их соединения для выполнения последовательностей действий.*

Линейная структура

Все блоки выполняются последовательно, стрелки идут в одном направлении. Используется для обычных вычислений.

- овал или скруглённый прямоугольник — **начало** и **конец**;
- параллелограмм — **ввод или вывод данных**;
- прямоугольник — **вычисление или действие**, обычно формула.

Условие и сравнение в прямоугольник не записывают.

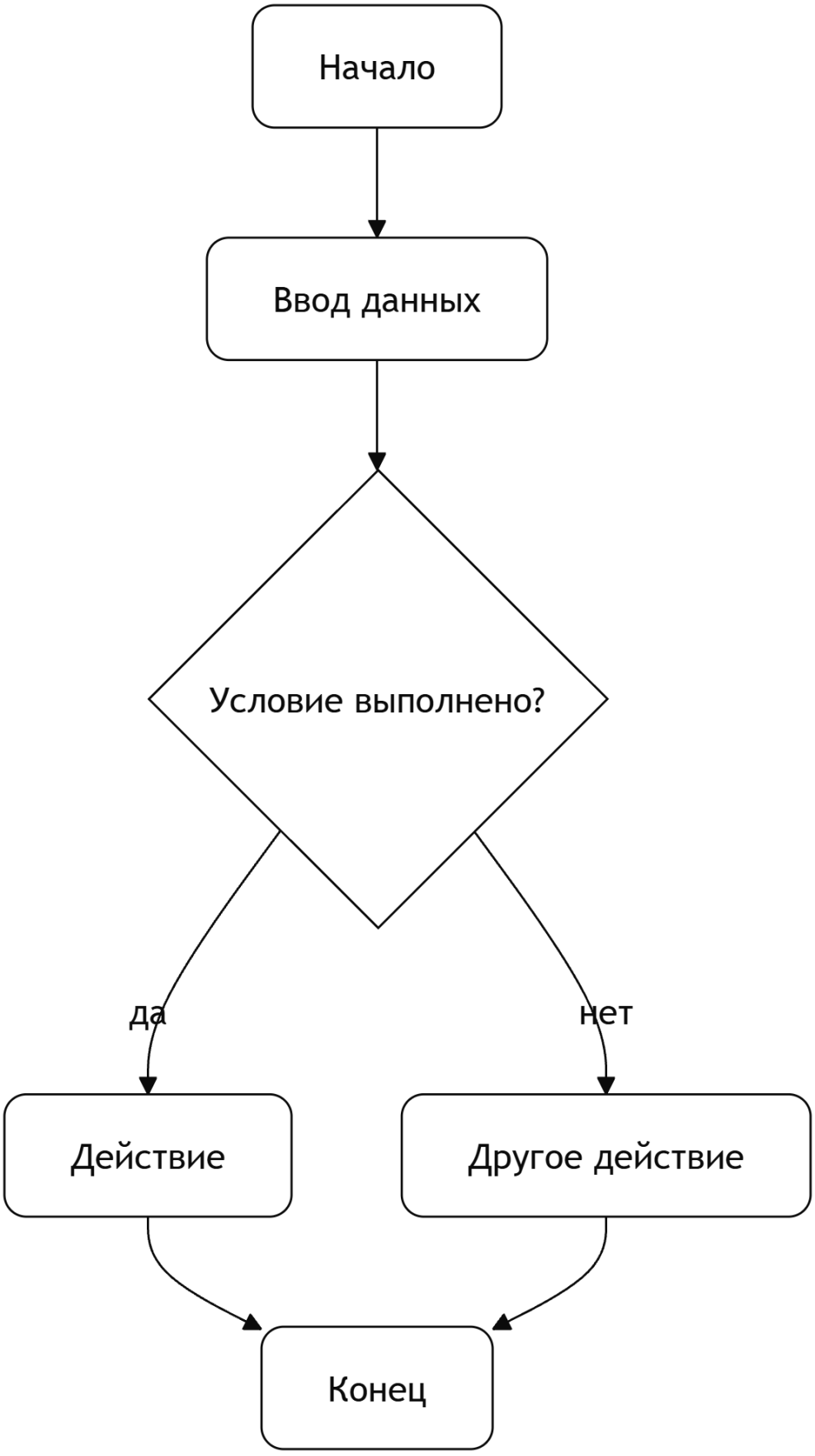
Ветвящаяся структура

Применяется, когда результат зависит от условия: выполняется одна ветвь или другая. Условие записывается в **ромбе**. В блок-схеме должны быть предусмотрены обе ветви — «да» и «нет», даже если по одной из них действие отсутствует. Варианты ветвления включают обход, выбор из двух вариантов и выбор из нескольких вариантов.

Циклическая структура

Обеспечивает многократное выполнение последовательности действий. Для цикла по условию используется ромб. Для цикла со счётчиком применяется специальный **шестиугольник**.

Сделано на сайте Зачёт | zachet.app



Блок-схему можно создавать в любом удобном редакторе или аккуратно рисовать вручную. Главное — корректность и читаемость.

Программа и языки

> ***Программа*** *— последовательность команд, предназначенных для выполнения задачи вычислительным устройством.*
> ***Команда*** *определяет один шаг процесса выполнения программы.*

Программное обеспечение делится на три класса:

- **системное** — обеспечивает работу технических компонентов компьютера;
- **прикладное** — решает пользовательские задачи;
- **инструментальное** — используется для разработки программ.

Языки программирования классифицируются по уровню близости к машинному коду:

Класс	Особенности
Машинные языки	Нули и единицы, понятные процессору; неудобны для человека, но позволяют напрямую управлять машиной.
Ассемблер	Символьное представление машинных команд ; относится к низкому уровню.
Языки высокого уровня	Более понятны человеку и предназначены для разных задач.

Универсального «лучшего» языка нет: один удобнее для визуализации, другой — для математических вычислений, третий — для обучения или конкретной платформы.

C и C++

Язык **C** развивался от ранних языков A и B. В 1979 году Бьёрн Страуструп начал развитие C++, расширив возможности C. Главное отличие C++ — поддержка **объектно-ориентированного программирования**: объектов, классов, наследования и инкапсуляции.

В задачах первого семестра используются преимущественно базовые возможности C и C++, поэтому объектно-ориентированные особенности пока несущественны.

Программы можно писать в разных средах, но проверять их следует в среде, поддерживающей стандарт языка. Рекомендуется **Visual Studio**: компилятор строго проверяет соответствие стандарту. Разные среды могут по-разному обрабатывать ошибки и нестандартные конструкции. Версия Visual Studio для учебных задач принципиального значения не имеет.

Структура программы C++

> *Программа на C++ — набор функций, описаний и директив препроцессора.*

Директивы препроцессора

Директива начинается с символа `#` и подключает заголовочные файлы или выполняет другую предварительную обработку. Библиотеки содержат готовые функции, например для ввода-вывода и математических операций.

Подключение записывают так:

- `#include <>` поиск в стандартных библиотеках;
- `#include ""` сначала поиск в текущем каталоге, затем в стандартных библиотеках.

Кавычки удобны для собственных заголовочных файлов.

Функции

Программа может содержать одну или несколько функций. Выполнение начинается с главной функции `main`.

Общий вид определения функции:

```
B8?0@0<5B@K
  ?5@0B>@K
}
```

- перед именем указывается тип возвращаемого значения;
- круглые скобки содержат параметры или остаются пустыми;
- фигурные скобки ограничивают тело функции;
- каждый оператор заканчивается точкой с запятой, а не обязательно каждая строка.

Описание сообщает компилятору о существовании функции, а определение содержит её реализацию. Подробно это различие рассматривается при изучении функций.

Главная функция часто объявляется как `int main()`: возвращаемое значение сообщает результат завершения программы.

Переменные и идентификаторы

Объявление переменной включает тип, имя и, при необходимости, начальное значение:

```
B8?O2K@065=85
```

Для нескольких объектов одного типа объявления можно объединять. Константа не может менять значение после инициализации. Вещественные числа в C++ записываются через точку, например `8.5`.

Идентификатор — имя переменной, константы или функции. Допустимы латинские буквы, цифры и знак подчёркивания. Нельзя:

- начинать имя с цифры;
- использовать пробелы;
- использовать точку;
- смешивать регистр без понимания последствий.

Прописные и строчные буквы различаются: `value` и `Value` — разные идентификаторы. Хотя некоторые компиляторы допускают русские буквы, рекомендуется использовать английские имена и понятный стиль записи.

Комментарии и область видимости

Комментарий — пояснение, игнорируемое компилятором. Он помогает отметить назначение переменной, функции или участка программы и временно исключить код из выполнения.

Однострочный комментарий:

```
// ?>OA=5=85
```

Многострочный комментарий:

```
/*  ?>OA=5=85
 2=5@Ab:>
```

Большую программу удобно проверять постепенно: сначала запустить небольшой рабочий фрагмент, затем добавлять следующий. Если после добавления появилась ошибка, проблемный участок легче найти.

Переменная, объявленная внутри функции или блока, доступна только там. **Глобальная переменная**, объявленная вне функций, доступна во всей программе, но глобальные переменные без необходимости использовать не следует: локальная область видимости делает код понятнее и безопаснее.

Структура программы может включать директивы, глобальные объявления и несколько функций, среди которых обязательно присутствует `main`.