



*Белорусский государственный университет
информатики и радиоэлектроники*

Основы конструирования программ

Преподаватель: Меженная Марина Михайловна

К.Т.Н., доцент,

доцент кафедры инженерной психологии и эргономики
а 609-2

mezhenная@bsuir.by



Кафедра инженерной психологии и эргономики

Структура курса «Основы конструирования программ»

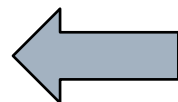
1 семестр: 4 лекции + практические занятия → зачет

2 семестр: курсовая работа



Где скачать материалы курса:

bit.ly/1Oow7ij



на моем гугл-диске

Google Диск

Меженная: Литература для студентов > Основы конструирования программ

Файлы



PDF 1 Навроцкий, А. А. О...



PDF 2 Шилдт. C++ базов...



PDF 3 Правила оформле...



PDF 4 Макконнелл. Сов...



PDF 5 Липпман C++ для ...



PDF 6 Культин. C,C++ в з...



Меженная М.М.

Где найти общую полезную информацию по курсу:

на моей персональной странице портала bsuir.by

ДИСЦИПЛИНА "ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ"

bit.ly/1jY3MaR

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ "ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ"

[ОКП. Методические указания к выполнению курсовой работы](#)

[ОКП Бланк задания на курсовую работу](#)

[ОКП Образец титульного листа](#)

[Стандарт 2017-ДП](#)

ЛИТЕРАТУРА ПО ДИСЦИПЛИНЕ "ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ"

[1 Навроцкий, А. А. Основы алгоритмизации и программирования в среде VISUAL C++ Уч. мет. пос.](#)

[2 Правила оформления кода на C++](#)

[3 Microsoft Visual Studio. Установка, запуск и отладка](#)

ПОЛЕЗНЫЕ ССЫЛКИ ПО ДИСЦИПЛИНЕ "ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ":

<http://purecodecpp.com/>

<http://ci-plus-plus-snachala.ru/>

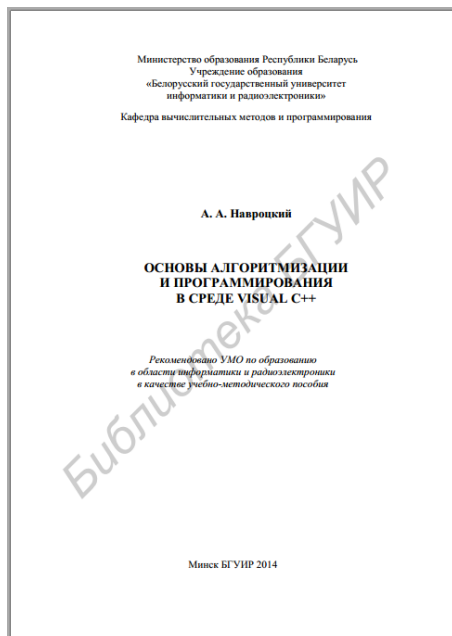
<http://cppstudio.com/>

[Векторный редактор диаграмм draw.io для разработки алгоритмов](#)

[ВОПРОСЫ К ЗАЧЕТУ ПО ДИСЦИПЛИНЕ "ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ"](#)

Литература по C++

+ интернет



А.А. Навроцкий Основы алгоритмизации и программирования в среде Visual C++

Герберт Шилдт C++ Базовый курс



Меженная М.М.

Литература по C++

Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Факультет компьютерного проектирования
Кафедра инженерной психологии и эргономики

М. М. Меженная

**ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ.
КУРСОВОЕ ПРОЕКТИРОВАНИЕ**

Рекомендовано УМО по образованию
в области информатики и радиоэлектроники для специальностей
1-40 05 01 «Информационные системы и технологии (по направлениям)»,
1-58 01 01 «Инженерно-психологическое обеспечение
информационных технологий»
в качестве пособия



Минск БГУИР 2019

М.М. Меженная Основы конструирования программ. Курсовое проектирование




Меженная М.М.

Литература из «Золотой» коллекции программиста



Стив МакКоннелл Совершенный код. Мастер-класс

Бесплатные видеоуроки Ерат, доступные (после регистрации) для всех желающих на *learn.by*

 **LEARN**
Digital Platform

CATALOGMY COURSES

SOFTWARE DEVELOPMENT METHODOLOGIES ⌚ COURSE IN PROGRESS: MAY 05, 2019 -	VERSION CONTROL WITH GIT ⌚ COURSE IN PROGRESS: MAR 17, 2019 -	MAVEN BUILD TOOL ⌚ COURSE IN PROGRESS: MAR 17, 2019 -
RUS CONTINUOUS INTEGRATION WITH JENKINS ⌚ COURSE IN PROGRESS: MAR 14, 2019 -	ENG APP MAKER ACADEMY ⌚ COURSE IN PROGRESS: MAR 01, 2019 -	ENG DIPLOMATIC ENGLISH ⌚ COURSE IN PROGRESS: JAN 11, 2019 -
RUS CLEAN CODE ⌚ COURSE IN PROGRESS: JAN 04, 2019 -	RUS WEBDRIVER ⌚ COURSE IN PROGRESS: JAN 01, 2019 -	ENG JAVA I/O ⌚ COURSE IN PROGRESS: JAN 01, 2019 -



Лекция 1: Введение. Конструирование в жизненном цикле разработки программного обеспечения. Программа и алгоритм.

План лекции:

1. Введение. Жизненный цикл разработки программного обеспечения (ПО).
2. Цель и задачи курса Основы конструирования программ.
3. Алгоритмические языки программирования. Программа и алгоритм.
4. Алгоритм: определение, свойства, способы описания, базовые конструкции. Зачем нужно уметь разрабатывать блок-схемы алгоритмов.



Жизненный цикл разработки ПО:

менеджер проекта

1.Идея создания ПО

2.Разработка требований

бизнес-аналитик

3.Проектирование:

- разработка архитектуры (высокоуровневое проектирование)
- проектирование пользовательского интерфейса

архитектор

UI, UX дизайнер

- детальное проектирование

4.Реализация (кодирование)

разработчик

5.Тестирование

Функциональный тестировщик

6.Выпуск и сопровождение

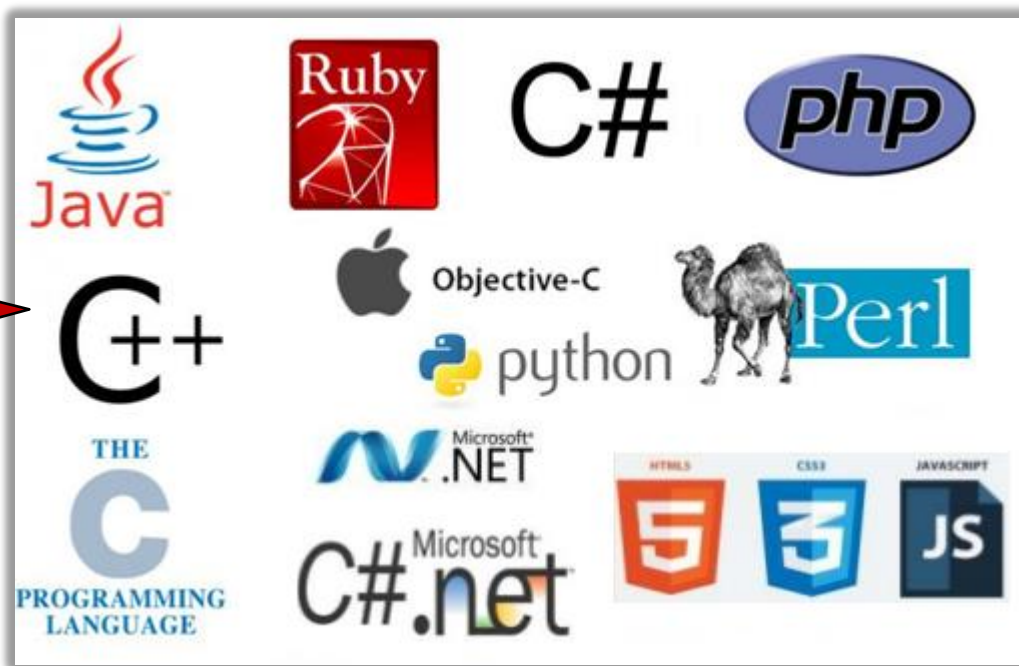
Специалист по автоматизации

DevOps



Жизненный цикл разработки ПО:

разработка



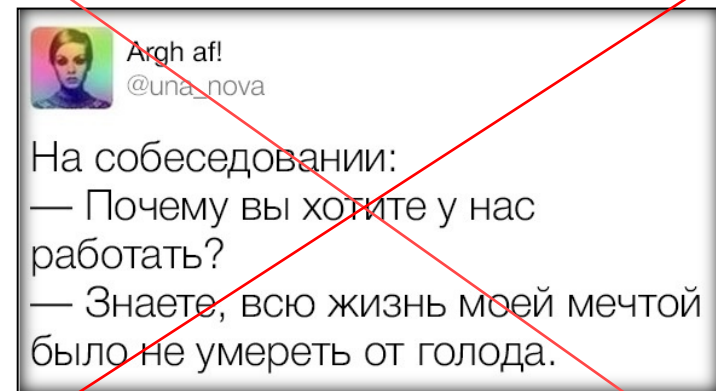
tiobe.com/tiobe-index/

Aug 2019	Aug 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.028%	-0.85%
2	2		C	15.154%	+0.19%
3	4	▲	Python	10.020%	+3.03%
4	3	▼	C++	6.057%	-1.41%
5	6	▲	C#	3.842%	+0.30%
6	5	▼	Visual Basic .NET	3.695%	-1.07%
7	8	▲	JavaScript		
8	7	▼	PHP		
9	14	▲▲	Objective-C		
10	9	▼	SQL		



На что обращает внимание работодатель:

- Технические навыки (ООП, базы данных, веб технологии, мобильная разработка, тестирование ПО, сетевые технологии)
- Английский язык (треб. уровень: B1 и выше)
- Личностные качества: адекватность, умение общаться, работать в команде
- Мотивационные качества (в том числе мотив вашего желания попасть на данную позицию)



Как попасть в IT или

Стратегия профессионального самоопределения

(! Информация сугубо рекомендательного характера)

1 курс: адаптация в университете

2 курс: учите английский язык

<http://www.park.by/>

<https://training.by/>

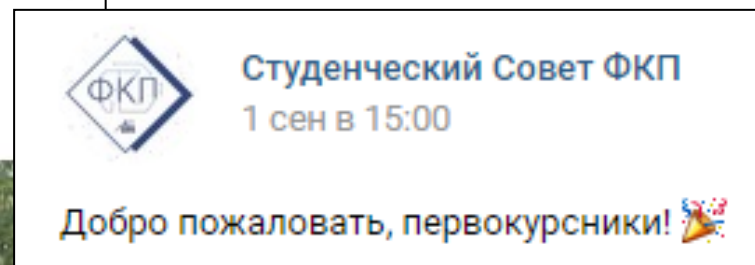
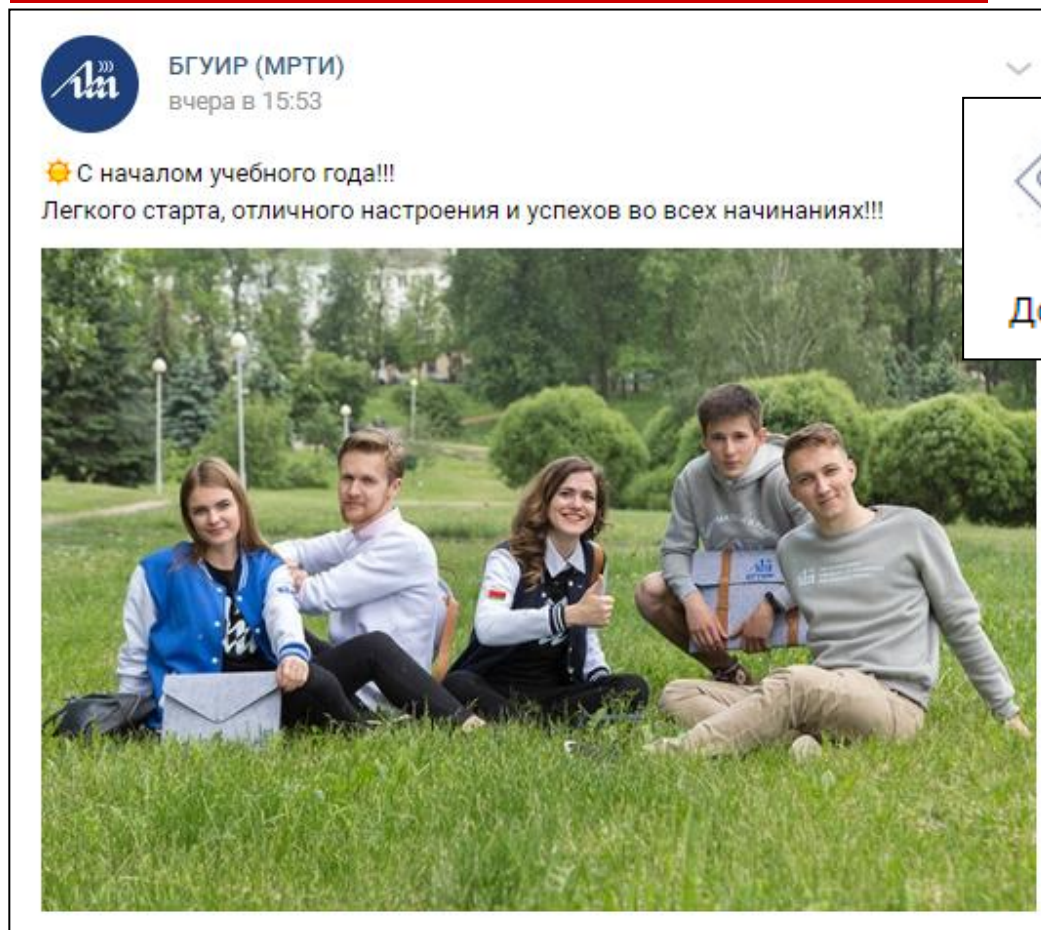
3 курс: выбор IT-профиля, отбор на тренинги, поиск работы

4 курс: распределение (! обязательно узнайте заранее, можно ли распределиться в конкретную негосударственную IT компанию)

P.S. Магистратура (1 год): аннулирует текущее распределение.



Паблики БГУИР и ФКП Вконтакте



ПОЧЕМУ ПАРК ВЫСОКИХ ТЕХНОЛОГИЙ?

Создавая благоприятные условия для развития ИТ-бизнеса, белорусский Парк высоких технологий является одним из крупнейших ИТ-кластеров в Центральной и Восточной Европе.

Уникальность ПВТ заключается в удачном сочетании качественного технического образования, высокого уровня профессионализма ИТ-специалистов и государственной поддержки ИТ-отрасли.

[Подробнее о ПВТ](#)

КАЛЕНДАРЬ

30 сентября 2019

Тестирование космического железа и ПО

24 сентября 2019

Юристы – фаундерам: как оформить команду и не спугнуть инвестора

17 сентября 2019

Автоматизация встроенных систем. Вхождение функциональщиком в автоматизацию

16 сентября 2019

Практический кейс построения финансовой модели ИТ-компании

11 сентября 2019

Пилим Legacy – дилемма бизнес-аналитика

[Все события](#)

Новости и события

21 мая 2019



[Справочник "ИТ-абитуриент 2019"](#)





[Подробнее](#)


















08 августа 2019

Резидент ПВТ «Гейм Стрим» за лето помог с лечением и отдыхом лямм сотням летей из японских

  TRAINING CENTER

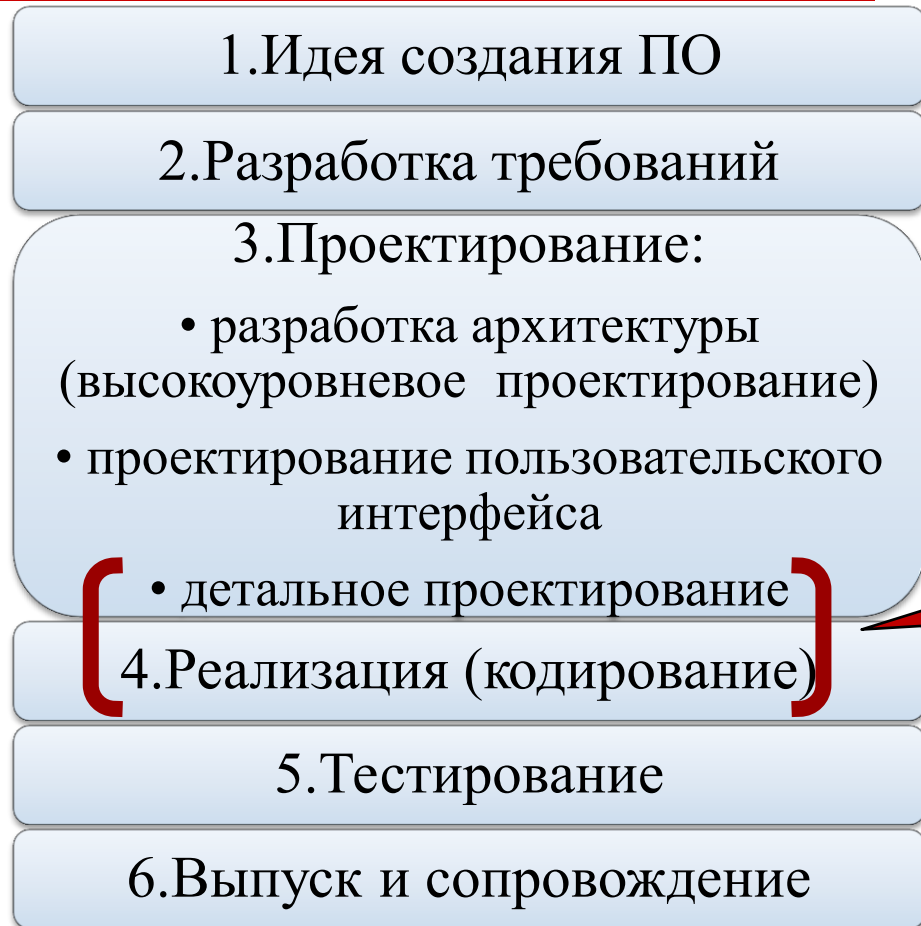
СПИСОК ТРЕНИНГОВ | О НАС | НОВОСТИ | FAQ | QUIZ

  ВОЙТИ

 AUTOMATED TESTING  Минск, Беларусь  2019-сент. Недель: ∞ РЕГИСТРАЦИЯ	 JAVASCRIPT DEVELOPMENT  Минск, Беларусь  2019-сент. Недель: 24 РЕГИСТРАЦИЯ	 CLOUD AND DEVOPS  Минск, Беларусь  2019-сент. Недель: 11 РЕГИСТРАЦИЯ	 INTRODUCTION TO JAVA ONLINE  Минск, Беларусь  2019-окт. Недель: ∞ РЕГИСТРАЦИЯ
		 	

?

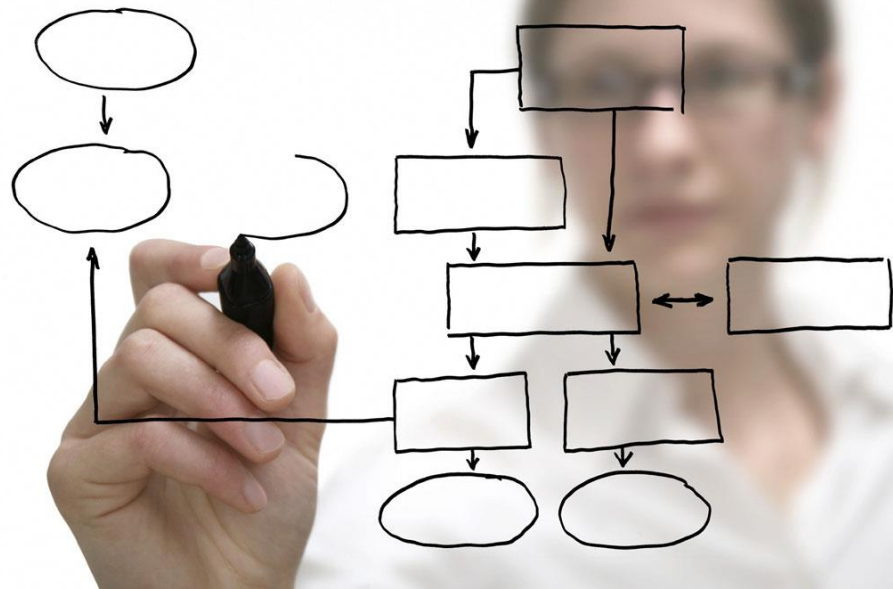
Конструирование в жизненном цикле разработки ПО:



конструирование

Конструирование ПО = детальное проектирование + программирование

Детальное проектирование – декомпозиция системы до уровня очевидно реализуемых модулей и функций, проектирование классов и методов.



Конструирование ПО = детальное проектирование + программирование

Программирование в части:

- выбор имен переменных и констант;
- выбор управляющих структур и организация блоков команд;
- «шлифовка» кода путем его форматирования и комментирования;
- интеграция программных компонентов, созданных по отдельности;
- оптимизация кода, направленная на повышение его быстродействия, и снижение степени использования ресурсов.



Конструирование ПО = детальное проектирование + программирование

В конструирование не входят такие процессы, как:
управление проектом,
разработка требований,
разработка архитектуры приложения,
проектирование пользовательского интерфейса,
тестирование системы и ее сопровождение.

Цель и задачи курса Основы конструирования программ

Целью настоящего курса является изучение принципов разработки программ и развитие базовых навыков культуры программирования.



Цель и задачи курса Основы конструирования программ

Задачи курса на уровне программирования:

- изучить базовые конструкции алгоритмического языка;
- научиться из этих базовых конструкций разрабатывать блок-схемы алгоритмов;
- научиться реализовывать эти алгоритмы на языке C++;
- освоить качественный стиль программирования, основанный на следовании соглашению о коде (Code Convention) и полезным эвристикам.

Задачи курса на уровне конструирования:

- изучить технологии конструирования программ (нисходящее/восходящее проектирование; структурное программирование/объектно-ориентированное программирование).

Цель и задачи курса Основы конструирования программ

Предусмотрена реализация полученных знаний и навыков в рамках курсовой работы.

Итогом изучения дисциплины является получение навыка построения небольших качественных программ на языке C++ в рамках парадигмы процедурного программирования.



Что останется за рамками этого конкретного курса:
Объектно-ориентированное программирование,
шаблоны проектирования,
UML-диаграммы.

Алгоритмические и неалгоритмические языки программирования

Алгоритмические языки программирования (императивные): описывается алгоритм решения задачи.

Примеры: Pascal, Delphi, C, C++, Java.

Неалгоритмические языки программирования (декларативные): описывается результат (его свойства), а не методы его достижения.

Примеры: Prolog, Lisp, Mercury.

Программа и алгоритм

Курс Основы конструирования программ в конечном итоге направлен на работу с алгоритмическими языками программирования.

Суть программирования на алгоритмическом языке состоит в том, чтобы научить исполнителя (компьютер) решать поставленную задачу. В этом смысле **программа представляет собой алгоритм, записанный на языке исполнителя.**

Алгоритм

Алгоритм – набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата.

Формальный исполнитель – субъект, механически реализующий алгоритм.

Совокупность допустимых действий образуют **систему команд исполнителя**.

Применительно к разработке ПО формальным исполнителем алгоритма выступает компьютер, а системой команд исполнителя является совокупность допустимых команд конкретного языка программирования.

Алгоритмизация – процесс разработки алгоритма.



Свойства алгоритма:

1.Понятность – каждая команда должна входить в систему команд исполнителя.

2.Дискретность – алгоритм должен представлять процесс решения задачи как последовательное выполнение шагов.

3.Однозначность – каждое правило алгоритма должно быть четким, однозначным.

4.Конечность – при точном исполнении всех предписаний алгоритма процесс должен прекратиться за конечное число шагов и при этом должен получиться определённый результат. Вывод о том, что решения не существует - тоже результат.

5.Универсальность – алгоритм должен быть применим к разным наборам исходных данных, которые составляют область применимости алгоритма.



Формы представления алгоритма

1. Словесная

Пример 1:

Записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел.

1. задать два числа;
2. если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3. определить большее из чисел;
4. заменить большее из чисел разностью большего и меньшего из чисел;
5. повторить алгоритм с шага 2.

Пример 2: метод «программирования комментариев».



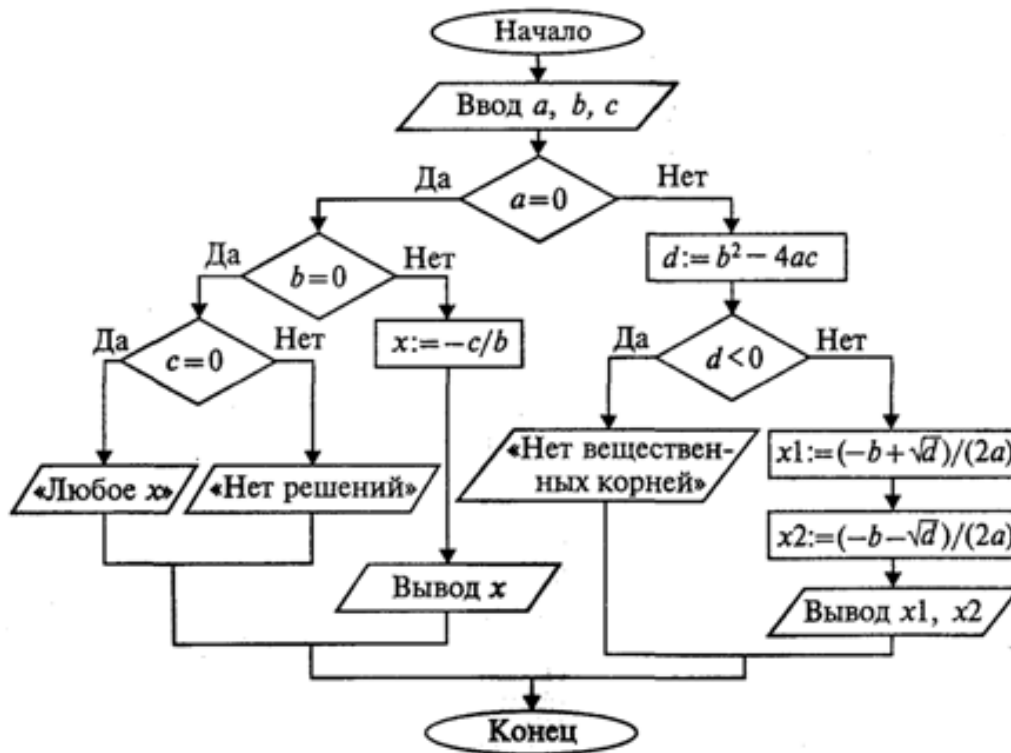
Формы представления алгоритма

2.Псевдокод (полуформализованное описание алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.)

```
процедура print_word(символный указатель w)
{
целое i = 0;
пока w[i] истинно (пока i элемент массива w существует)
вывести на экран w[i++](следующий элемент w);
}
```

Формы представления алгоритма

3. Графическая (блок-схемы)



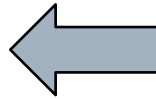
Формы представления алгоритма

4. Программная (текст на языке программирования).

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, world!" << endl;
    system("pause");






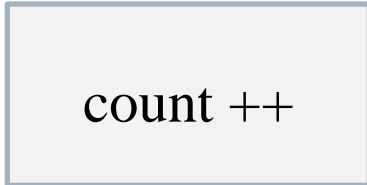
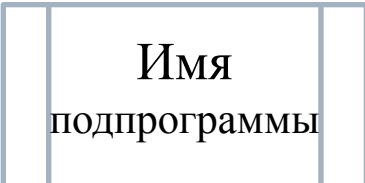

    return 0;
}
```




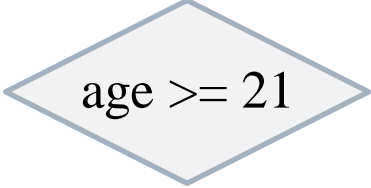

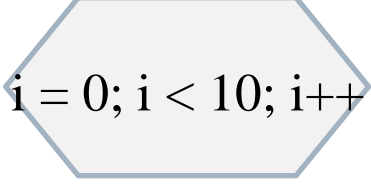
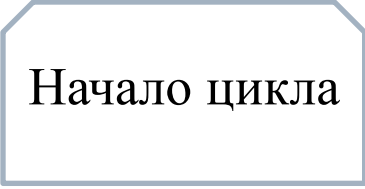
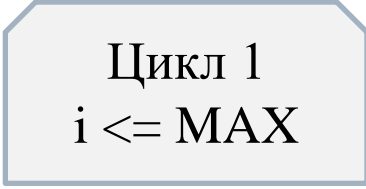
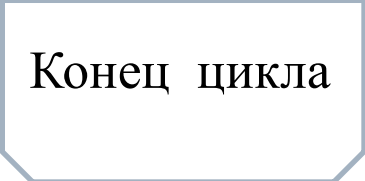

Программа представляет собой алгоритм, реализованный посредством конкретного языка программирования.

В этом смысле понятие алгоритма шире понятия программы.

Основные символы блок-схемы алгоритма

 <p>Начало/Конец</p>	Начало, конец алгоритма, возврат из функции/метода	 <p>Начало</p>
 <p>Ввод/вывод данных</p>	Ввод или вывод данных	 <p>Ввод count</p>
 <p>Действие</p>	Функциональный блок	 <p>count ++</p>
 <p>Имя подпрограммы</p>	Вызов предопределенного процесса (функции/метода)	 <p>readFile</p>

Основные символы блок-схемы алгоритма

 Условие	Логический блок (для реализации условного оператора if)	 age >= 21
 Параметр цикла	Блок модификации (для реализации цикла for)	 i = 0; i < 10; i++
 Начало цикла	Блоки границ цикла (для реализации циклов while, do-while, допускается в том числе и для цикла for)	 Цикл 1 i <= MAX
 Конец цикла		 Цикл 1

Основные символы блок-схемы алгоритма

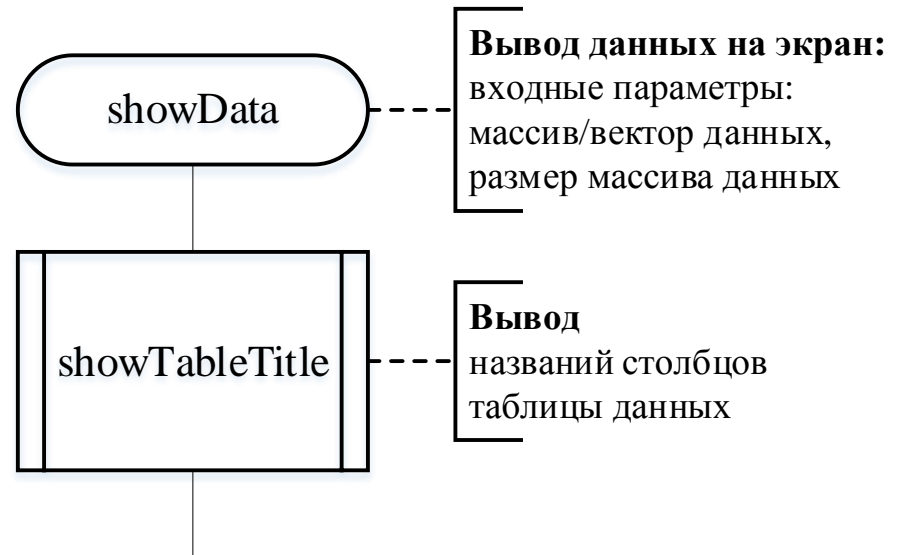


Соединитель для указания связи между прерванными линиями схемы. Должен содержать уникальное обозначение.

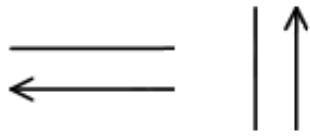


Основные символы блок-схемы алгоритма

-- [Комментарий для добавления пояснительных записей, в том числе когда объём текста, помещаемого внутри некоего символа, превышает его размер. Комментарии часто используют для описания входных аргументов функции/метода.



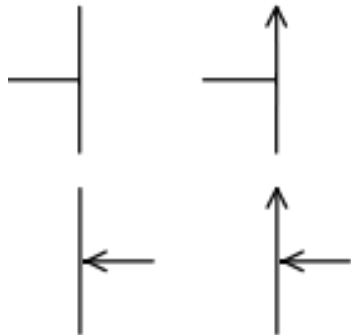
Обозначение соединений между символами блок-схемы алгоритма



Указание направления линии потока:
без стрелки, если линия направлена слева направо или сверху вниз;
со стрелкой в остальных случаях



Изменение направление потока (под углом 90°)



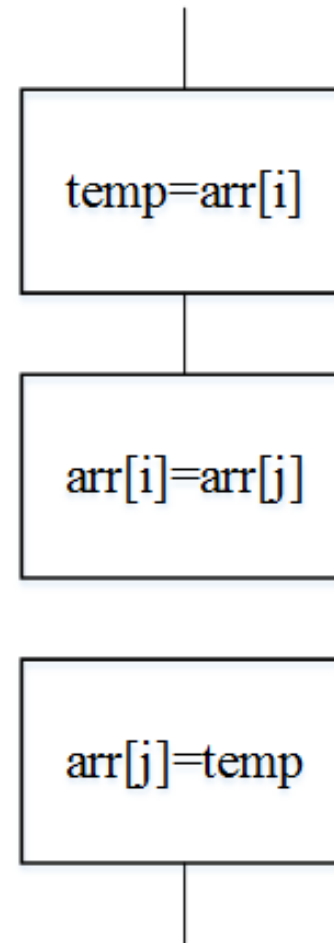
Слияние линий потока, каждая из которых направлена к одному и тому же символу на схеме

Пересечение двух НЕсвязанных потоков следует избегать!

Базовые алгоритмические структуры: следование

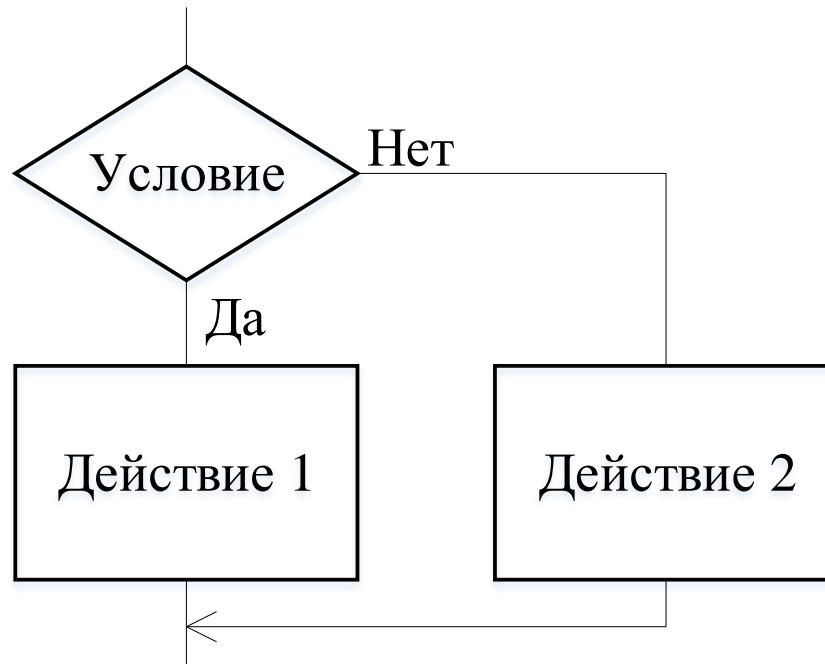


Пример:
меняем местами два
элемента массива



Базовые алгоритмические структуры: ветвление (полное)

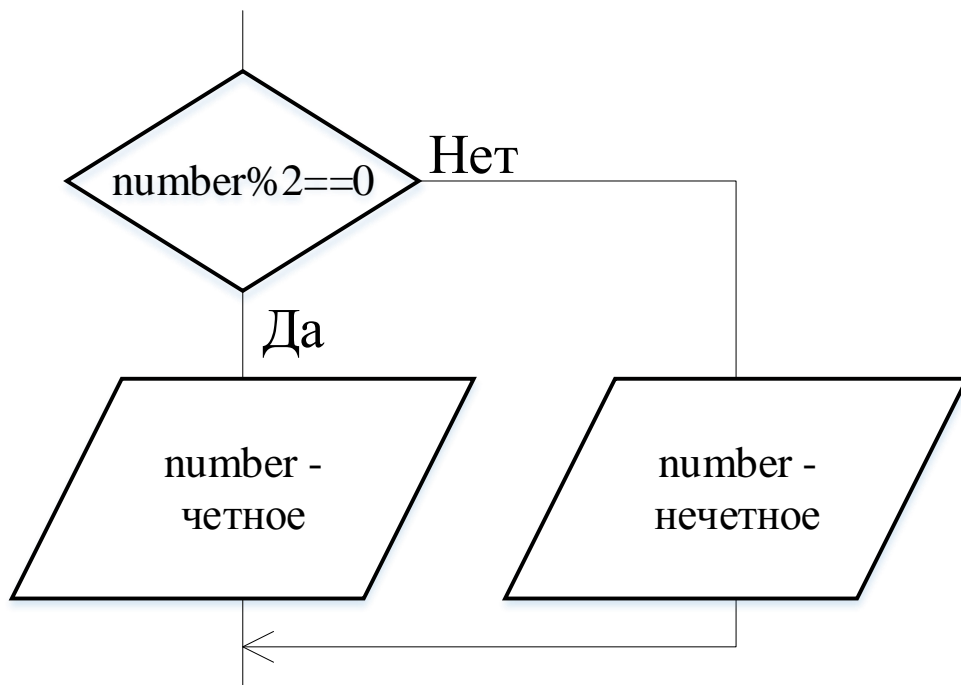
Реализуется посредством оператора if-else



Базовые алгоритмические структуры: ветвление (полное)

Реализуется посредством оператора if-else

Пример: определить четное или нечетное число.



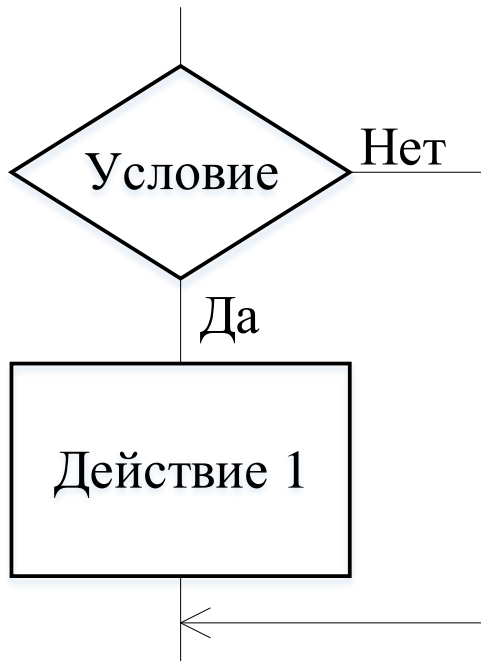
// проверка на четность

```
int number;  
cout << "Enter number: " << endl;  
cin >> number;
```

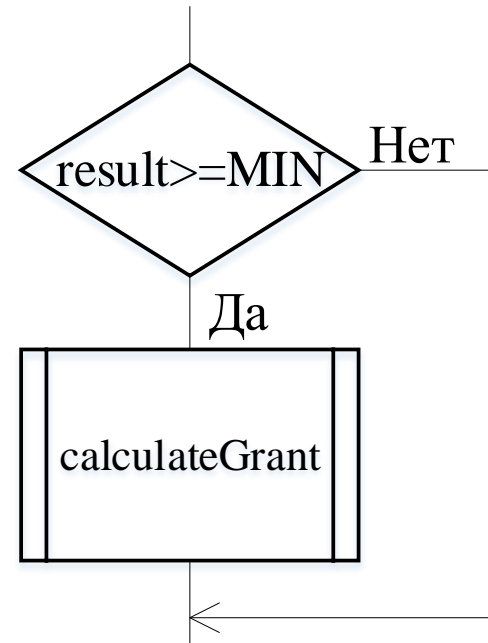
```
if (number % 2 == 0)  
{  
    cout << "Четное" << endl;  
} else  
{  
    cout << "Нечетное" << endl;  
}
```


Базовые алгоритмические структуры: ветвление (неполное)

Реализуется посредством оператора if

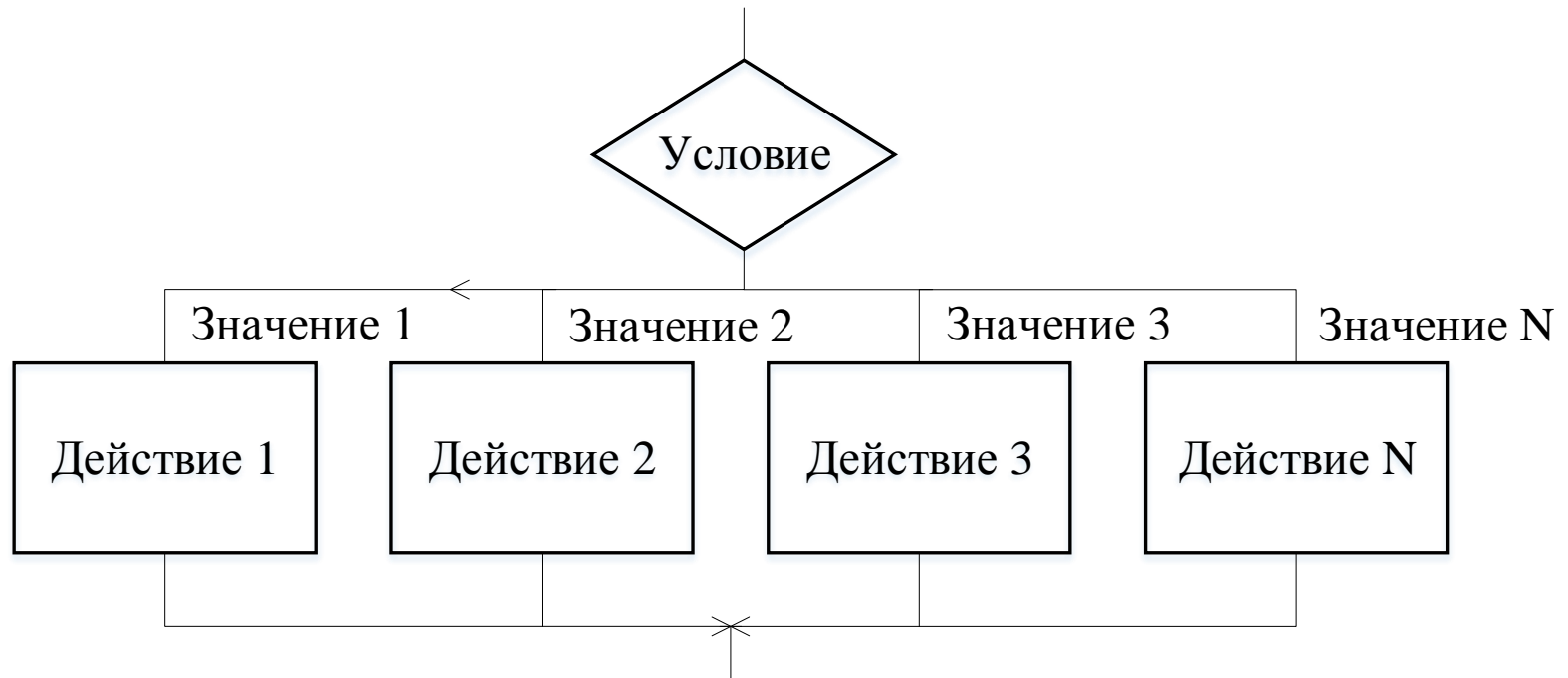


Пример: начислить стипендию, если студент набрал достаточно баллов за сессию



Базовые алгоритмические структуры: ветвление (выбор)

Реализуется посредством оператора switch-case



Базовые алгоритмические структуры: ветвление (выбор)

Реализуется посредством оператора switch-case

Пример: по номеру месяца вывести его название

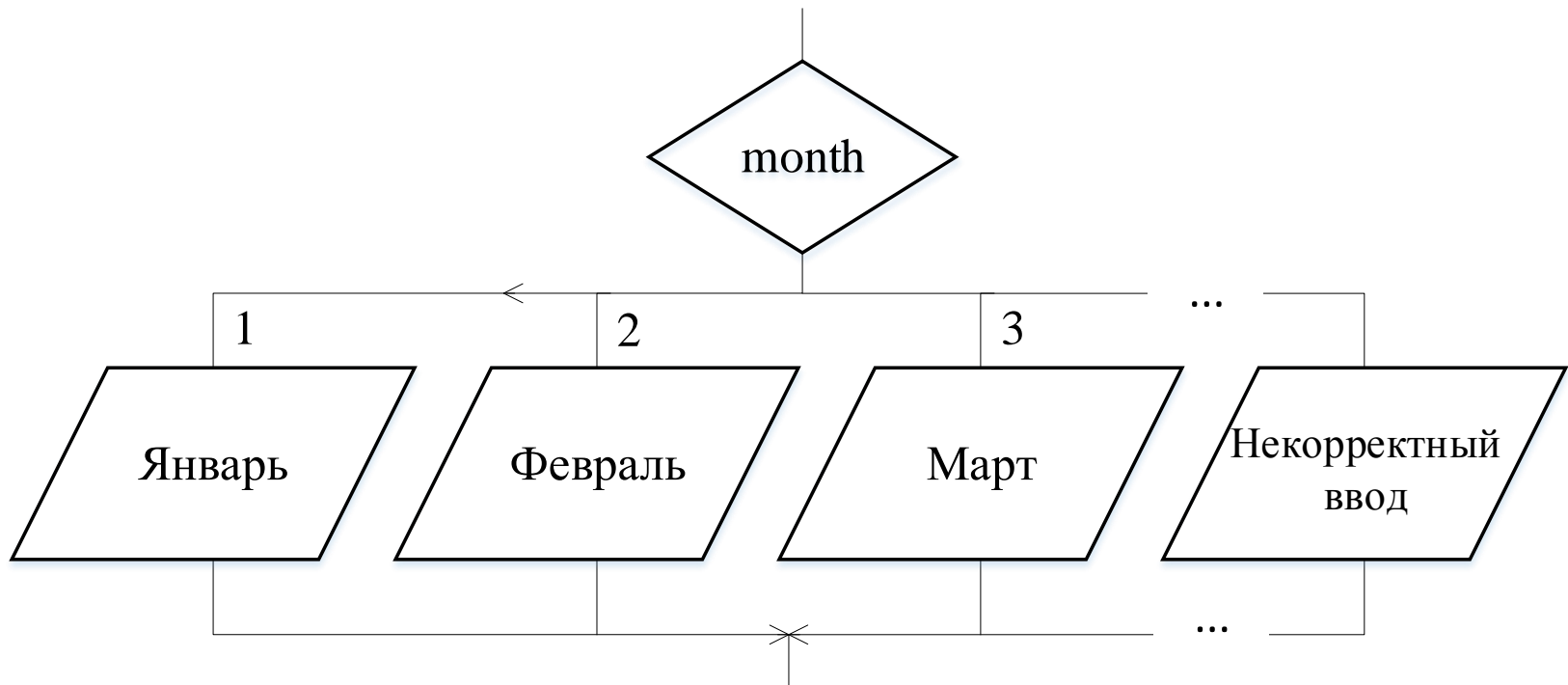
```
switch (month) {  
    case 1:  
        cout << "Январь" << endl;  
        break;  
    case 2:  
        cout << «Февраль" << endl;  
        break;  
    ...  
    case 12:  
        cout << "Декабрь" << endl;  
        break;  
    default:  
        cout << "Введите корректные данные!" << endl;  
        break;  
}
```



Базовые алгоритмические структуры: ветвление (выбор)

Реализуется посредством оператора switch-case

Пример: по номеру месяца вывести его название



Базовые алгоритмические структуры: цикл (с предусловием)

Реализуется посредством оператора while



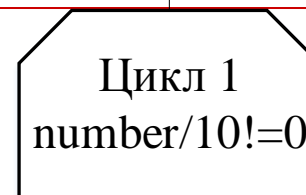
Тело цикла

Цикл 1

Пример: определить количество цифр числа

count = 1;

```
while (number / 10 != 0)
{
    number = number / 10;
    count++;
}
```



number/=10

count++

Цикл 1

Базовые алгоритмические структуры: цикл (с постусловием)

Реализуется посредством оператора do-while

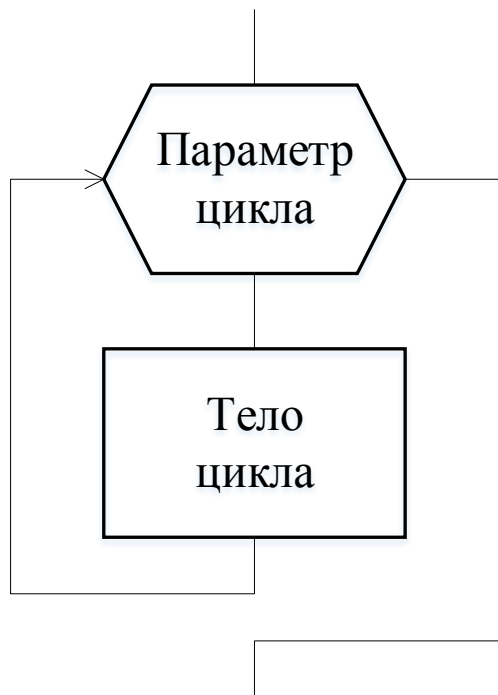


Пример:
реализовать
игру в карты
между
пользователем и
компьютером
(игра
продолжается,
пока не найден
победитель)



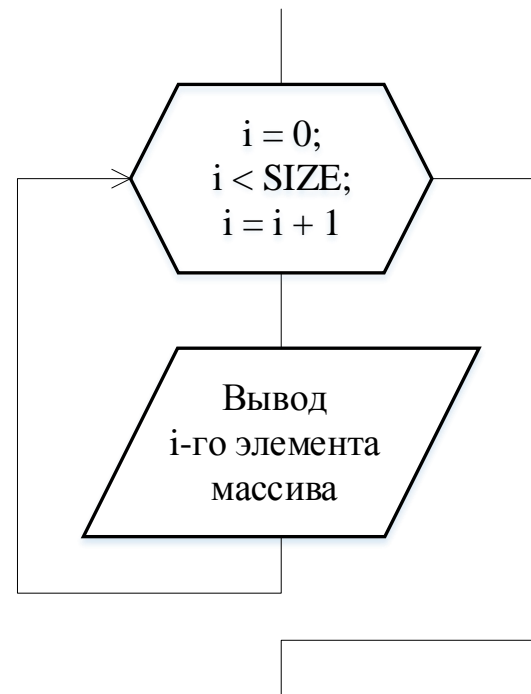
Базовые алгоритмические структуры: цикл (с предусловием)

Реализуется посредством оператора for



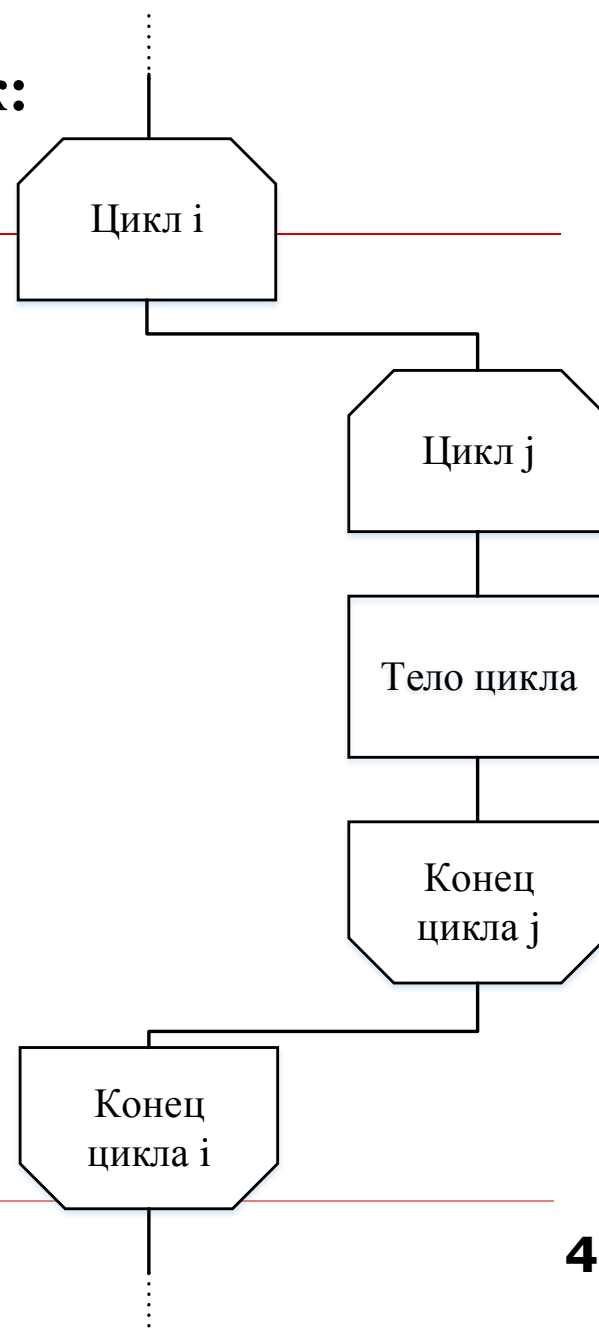
Пример: вывести все элементы массива

```
for (int i = 0; i < SIZE; i++)  
{  
    cout << arr[i] << " ";  
}
```



Цикл for может быть реализован и так:

Такое представление цикла for более читабельное (особенно, если, несколько циклов for вложены друг в друга) нежели предыдущий вариант с «обвязкой»



Базовые алгоритмические структуры

Обратите внимание, что каждая базовая структура в конечном итоге имеет один вход и один выход!



Зачем нужно разрабатывать блок-схемы алгоритмов, если можно сразу написать код (или зачем нужны алгоритмы)?

- В обучающих целях (визуализация базовых алгоритмических конструкций, составляющих основу любого алгоритмического языка программирования; развитие логического мышления)
- Это универсальный общепринятый инструмент создания любого рода инструкций (например, руководство пользователя, медицинские методики и др.)
- Для решения сложных задач (когда сразу написать код не получается, или когда требуется реализовать известное решение, представленное в виде алгоритма).

Зачем нужно разрабатывать блок-схемы алгоритмов, если можно сразу написать код (или зачем нужны алгоритмы)?

- А еще вы будете рисовать алгоритмы в рамках курсового проектирования(включая курсовую по ОКП) и дипломного проектирования

