Модуль 1.1. Введение

0,5 час

Цели и задачи дисциплины

- Учебная дисциплина «Программирование» является общепрофессиональной дисциплиной и входит в перечень обязательных дисциплин, включенных в учебный план вузом
- *Целью* дисциплины является формирование, развитие и становление у бакалавра следующих основных общекультурных и профессиональных компетенций: владение культурой мышления, способность к обобщению, анализу, восприятию информации, постановке цели и выбору путей ее достижения; умение понять поставленную задачу; умение самостоятельно разработать алгоритм решения задачи; умение реализовать алгоритм на изучаемом языке программирования высокого уровня; владение современными средами разработки программного обеспечения
- Основная задача дисциплины состоит в том, чтобы сформировать способности:
  1) выполнять анализ предметной области прикладной задачи, находить методы ее решения, выполнять формальную постановку задачи;
  2) разрабатывать алгоритмы решения задачи и записывать их различными способами;
  3) создавать программы на изучаемом языке программирования;
  4) использовать современные средства разработки программного обеспечения.

Краткая характеристика дисциплины, её место в учебном процессе

- Бакалавр по направлениям 231000 – Программная инженерия, 230100 – Информатика и вычислительная техника и 010500 – Математическое обеспечение и администрирование ИС должен владеть базовыми знаниями в области алгоритмизации и программирования. Под этим понимается системный подход к решению задач, алгоритмическое мышление, знание терминологии и современных средств разработки программного обеспечения. В настоящее время существует большое количество разнообразных языков программирования, с помощью которых можно эффективно решать широкий круг задач. Но залогом успешной разработки программного обеспечения на любом языке программирования было и остается знание основных принципов алгоритмизации, понимание процесса работы программы, обработки компьютером данных. Поэтому особое внимание при изучении курса уделяется различным способам организации данных в программе, решению стандартных алгоритмических задач.
- Освоение основ алгоритмизации и программирования построено на изучении практических приемов программирования на языках С и С ++ в среде программирования Microsoft Visual Studio. Но, несмотря на это, все изучаемые темы распространяются на любые языки программирования высокого уровня.

Краткая характеристика дисциплины, её место в учебном процессе

- Язык С выбран из-за стабильности языка и его окружения (стандартные библиотеки, компиляторы и другие инструментальные средства), а также наличия возможности получения программ, выполняющихся с максимальной скоростью на данной аппаратной платформе
- Компиляторы, библиотеки и инструменты разработки на языке С существуют практически для всех систем
- Программы на языке С отличаются переносимостью между платформами на уровне исходного кода
- Синтаксис многих инструкций языка С лежит в основе таких языков, как C++, C#, Java, PHP

- Настоящая программа учебной дисциплины рассчитана на **216** часов занятий, в том числе 96 часов отводится на аудиторные занятия и 120 часов на самостоятельную работу студентов.
- Курс предполагает лекционные и лабораторные занятия, а также выполнение курсовой работы/проекта

Распределение учебного времени дисциплины «Программирование»

Форма обучения	Семестр и его продолжительность (нед.)	РАСПРЕДЕЛЕНИЕ							
		Общей трудоемкости (час/ЗЕТ)	В том числе					Форм СРС	Форм ПА - аттестация
			На аудиторные занятия (час)				на СРС (час)		
			Всего / в интерактивной форме	В том числе					
				Л/ в интерактивной форме	Лб/ в интерактивной форме	Пр.(С)			
1	2	3	4	5	6	7	8	9	10
очная	1 год, 5 блок	216 / 6	96 / 24	32/6	64 / 18	-	120	КП КР ИЗЛР	-
Всего по очной форме обучения		216 / 6	96 / 24	32/6	64 / 18	-	120		

Связь с другими дисциплинами

Связь с предшествующими дисциплинами

Для формирования общекультурных и профессиональных компетенций необходимы базовые компетенции, сформированные при изучении учебных дисциплин учебного плана «Информатика», «Архитектура вычислительных систем», «Математический анализ».

Для успешного изучения дисциплины студент должен:

- знать основные принципы организации и функционирования современного компьютера; вид представления информации различного рода в памяти современного компьютера; позиционные системы счисления, способы перевода чисел из одной системы в другую; основы математического анализа;
- уметь: применять математические методы и вычислительные алгоритмы для решения практических задач, проектировать эксперимент и анализировать результаты; производить переводы чисел из одной системы счисления в другую; выполнять арифметические операции над двоичными и шестнадцатеричными числами;
- владеть: методами построения математической модели задач и содержательной интерпретации полученных результатов; основными методами, способами и средствами получения, хранения, переработки информации, навыками работы с компьютером как средством управления информацией.

Кроме того, знание английского языка облегчит усвоение среды программирования.

Связь с последующими дисциплинами

- Компетенции, сформированные в результате освоения содержания дисциплины «Программирование», необходимы для освоения большинства дисциплин профессионального цикла учебного плана. Практические навыки программирования необходимы при решении различных задач на ЭВМ.

Программирование как научная дисциплина

- *Программирование* – процесс и искусство создания компьютерных программ и/или программного обеспечения с помощью языков программирования. Программирование сочетает в себе элементы искусства, фундаментальных наук (прежде всего информатика и математика), инженерии, спорта и ремесла.
- В узком смысле слова, программирование рассматривается как кодирование алгоритмов на заданном языке программирования. Под программированием также может пониматься разработка логической схемы для программируемых логических интегральных схем (ПЛИС), а также процесс записи информации в постоянном запоминающем устройстве (ПЗУ).
- В более широком смысле программирование – процесс создания программ, то есть разработка программного обеспечения, которая включает в себя:
  1) анализ
  2) проектирование
  3) кодирование и компиляцию
  4) тестирование и отладку
  5) испытания и сдачу программ
  6) сопровождение

Модуль 1.2. Основные принципы алгоритмизации

1 час

Этапы создания программного обеспечения

В процессе разработки программ с использованием процедурного подхода выделяют следующие этапы:

- постановка задачи – определение требований к программному продукту
- анализ – формальная постановка задачи и определение методов ее решения
- проектирование – разработка структуры программного продукта, выбор структур для хранения данных, построение и оценка алгоритмов подпрограмм и определение особенностей взаимодействия с вычислительной средой
- реализация – кодирование алгоритма с помощью выбранного языка программирования, тестирование и отладка программы
- модификация – выпуск новых версий программного продукта

Постановка задачи: понятие задачи

Процесс создания нового программного обеспечения (ПО) начинается с постановки задачи, в процессе которой определяют требования к программному продукту (ПП)

Под задачей понимается проблемная ситуация с явно заданной целью, которую необходимо достичь

В узком смысле *задача*

- осознанная проблемная ситуация с выделенными условиями (данным) и требованием (целью)
- ситуация с известным начальным состоянием системы и конечным состоянием системы, причём алгоритм достижения конечного состояния от начального известен (в отличие от *проблемы*, в случае которой алгоритм достижения конечного состояния системы не известен).

В более широком смысле под *задачей* также понимается то, что нужно выполнить — всякое задание, поручение, дело, — даже при отсутствии каких бы то ни было затруднений или препятствий в выполнении.

В учебной практике понятие «*задача*», приобретает более узкий смысл и обозначает упражнение, требующее нахождения решения по известным данным с помощью определённых действий (умозаключения, вычисления, перемещения элементов и т. п.) при соблюдении определённых правил совершения этих действий

Постановка задачи

- *Постановка задачи* – точная формулировка условий задачи с описанием входной и выходной информации.

При постановке задачи

- устанавливается набор выполняемых функций, а также перечень и характеристики исходных данных. Например, для числовых данных может задаваться точность, для текстовых – размер текста, способ кодирования и т.п.
- определяется перечень результатов, их характеристики и способы представления. Выходная информация может быть представлена в виде документов, кадров на экране монитора, информации в базе данных, выходного сигнала устройству управления.
- определяется среда функционирования ПП: конкретную комплектацию и параметры технических средств, версию используемой операционной системы, параметры и характеристики программного обеспечения, с которым возможно предстоит взаимодействовать разрабатываемому ПО.

Постановкой задачи занимается заказчик задачи или консалтинговая компания. В результате согласования между заказчиком и исполнителем всех перечисленных вопросов оставляют техническое задание, которое служит основанием для дальнейшей работы.

Анализ, формальная постановка и выбор метода решения

- На данном этапе производится анализ предметной области задачи – части реального мира, рассматриваемой в пределах данного контекста. Под контекстом здесь может пониматься, например, область исследования или область, которая является объектом некоторой деятельности. Предметную область задачи составляют множество объектов, рассматриваемых в пределах данного контекста, а также свойства, отношения и функции, с помощью которых достигается выполнение поставленной цели.
- По результатам анализа выбирают математические абстракции, адекватно, то есть с требуемой точностью и полнотой, представляющие исходные данные и результаты. Строят формальную модель задачи и определяют метод решения задачи – метод преобразования исходных данных в результат.
- Формальная модель воплощается с помощью одного или нескольких формальных языков (например, языков математических теорий или языков программирования).

Анализ, формальная постановка и выбор метода решения: пример

Постановка задачи

Разработать программу, которая по заданным длинам сторон прямоугольника определяет его площадь.

- Исходными данными являются длины сторон, т.е. некоторые числовые значения, для которых должны быть заданы диапазон изменения и точность. Математические абстракции для представления исходных данных – некие изменяемые значения – переменные. Результат – площадь – также некоторое числовое значение, диапазон возможных значений и точность которого зависят от соответствующих характеристик исходных данных.

Формальная (математическая) постановка задачи

Входные данные: a, b – длины сторон прямоугольника, выраженные положительным целым числом

Выходные данные: S – площадь, целое число

Метод решения: Площадь вычисляется по формуле $S=a*b$

Анализ, формальная постановка и выбор метода решения

- Часто формальная постановка задачи однозначно определяет метод решения задачи. В тех случаях, когда задача может быть решена несколькими методами, выбирается один из них с учетом сложности и эффективности его реализации
- При использовании процедурного подхода сложные задачи в процессе анализа разбиваются на подзадачи, для каждой из которых может строиться своя модель и выбираться метод решения. При этом результаты решения одной задачи могут использоваться в качестве исходных данных в другой
- Определив методы решения, следует для некоторых вариантов исходных данных вручную вычислить ожидаемый результат. Это позволит более четко уяснить последовательность действий, что облегчит процесс разработки алгоритма. Кроме того, необходимо продумать для каких данных результат может не существовать

Проектирование (разработка алгоритма)

- Различают логическое и физическое проектирование
- Логическое проектирование не учитывает особенностей среды, в которой будет функционировать программа
- При физическом проектировании все эти факторы (технические и программные средства компьютера) наоборот должны быть учтены

Логическое проектирование

- Логическое проектирование при процедурном подходе предполагает детальную проработку последовательности действий будущей программы. Сначала определяется структура ПП: будет это отдельная программа или комплекс взаимосвязанных программ. Затем производиться разработка алгоритма.

Алгоритм

- Под *алгоритмом* понимают формально описанную последовательность действий, которые необходимо выполнить исполнителю для достижения указанной цели и получения требуемого результата.
- Алгоритм записывается на формальном языке, исключающем неоднозначность толкования. Исполнитель – это человек, компьютер, автоматическое устройство и т.п. Он должен уметь выполнять все команды, составляющие алгоритм, причем механически, "не раздумывая".
- Слово алгоритм происходит от *algorithmi* – латинской формы написания имени великого математика IX в. Аль Хорезми, который сформулировал правила выполнения арифметических действий. Первоначально под алгоритмами и понимали только правила выполнения четырех арифметических действий над многозначными числами. В дальнейшем это понятие стали использовать вообще для обозначения последовательности действий, приводящих к решению поставленной задачи.

Пример алгоритма

Рассмотрим пример алгоритма для нахождения середины отрезка при помощи циркуля и линейки

Алгоритм деления отрезка AB пополам:

1. поставить ножку циркуля в точку A
2. установить раствор циркуля равным длине отрезка AB
3. провести окружность
4. поставить ножку циркуля в точку B
5. провести окружность
6. через точки пересечения окружностей провести прямую
7. отметить точку пересечения этой прямой с отрезком AB

Анализ алгоритмов

- Анализ примеров различных алгоритмов показывает, что запись алгоритма распадается на отдельные указания исполнителю выполнить некоторое законченное действие. Каждое такое указание называется *командой*. Команды алгоритма выполняются одна за другой. После каждого шага исполнения алгоритма точно известно, какая команда должна выполняться следующей. Совокупность команд, которые могут быть выполнены исполнителем, называется системой команд исполнителя
- Таким образом, выполняя алгоритм, исполнитель может не вникать в смысл того, что он делает, и вместе с тем получать нужный результат. В таком случае говорят, что исполнитель действует формально, т.е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции

Свойства алгоритма

- *Детерминированность* (определенность или точность). Предполагает получение однозначного результата вычислительного процесса при заданных исходных данных. Благодаря этому свойству процесс выполнения алгоритма носит механический характер. (Например, в алгоритме указано, что надо взять 3—4 стакана муки. Какие стаканы, что значит 3—4, какой муки?).
- *Результативность*. Указывает на наличие таких исходных данных, для которых реализуемый по заданному алгоритму вычислительный процесс должен через конечное число шагов остановиться и выдать искомый результат.
- *Массовость* (универсальность). Это свойство предполагает, что алгоритм должен быть пригоден для решения всех задач данного типа. То есть алгоритм должен быть составлен так, чтобы им мог воспользоваться любой исполнитель для решения аналогичной задачи. (Например, правила сложения и умножения чисел годятся для любых чисел, а не для каких-то конкретных).
- *Дискретность*. Означает расчлененность определяемого алгоритмом вычислительного процесса на отдельные этапы, возможность выполнения которых исполнителем (компьютером) не вызывает сомнений.

Пример анализа алгоритма

Постановка задачи:

Разработать алгоритм нахождения наименьшего простого делителя натурального числа *k*, большего единицы

Формальная постановка задачи:

Входные данные: k – натуральное число, $k>1$;

Выходные данные: i – наименьший простой делитель числа *k*, $i \neq 1$.

Метод решения:

- Простым называется число, не имеющее делителей, отличных от единицы и его самого, причем единица во множество простых чисел не входит

$i := 3$

Анализ алгоритма из примера

Алгоритм решения задачи:

1: Положить целое число *i* равным двум и перейти на шаг 2.

2: Если *k* делится нацело на *i*, то завершить работу алгоритма, выдав в качестве результата *i*, иначе перейти на шаг 3.

3: Увеличить значение *i* на единицу и перейти на шаг 2.

Для того чтобы понять этот алгоритм, надо выступить в роли компьютера (или скорее даже универсального исполнителя команд), выполняя вручную указанную в нем последовательность действий для некоторых небольших значений *k*. Будем записывать значения величины *i* после каждого шага алгоритма

***k*= 2**	***k*= 3**	***k*= 4**
1: i=2	1: i=2	1: i=2
2: i=2	2: i=2	2: i=2
	3: i=3	
	2: i=3	

Рассмотрим, удовлетворяет ли алгоритм рассмотренным выше свойствам.

- *Детерминированность.* Действия, которые необходимо произвести на каждом шаге, должны быть определены строго и недвусмысленно в каждом возможном случае. В данном примере применена достаточно определенная, хотя и не вполне формальная система обозначений.
- *Конечность.* Алгоритм должен всегда заканчиваться после выполнения конечного числа шагов. Алгоритм удовлетворяет этому условию, так как величина *i* вначале меньше *k*, ее значение увеличивается на единицу к каждому очередному выполнению шага 2, и поэтому выполнение алгоритма будет прекращено на шаге 2 при *i*=*k*, если *k* – простое число, или ранее при составном *k*.
- *Массовость* (универсальность). Этот алгоритм пригоден для нахождения наименьшего простого делителя любого натурального числа *k*, большего единицы, а не для какого-то конкретного.
- *Дискретность*. Процесс нахождения наименьшего простого делителя разбит на отдельные этапы, возможность выполнения которых исполнителем (компьютером) не вызывает сомнений.

Разработка алгоритма методом пошаговой детализации

Создание программы - процесс сложный, поэтому практически с любого этапа возможен возврат на предыдущие этапы для исправления ошибок или принятия других проектных решений. Чаще всего такого рода возвраты являются следствием ошибок, допущенных при логическом проектировании программы. Поэтому в процессе программирования необходимо особое внимание уделять разработке алгоритмов.

Для разработки алгоритмов часто используют метод пошаговой детализации, согласно которому разработку выполняют поэтапно.

- На первом этапе описывают решение поставленной задачи, выделяя подзадачи и считая их решенными
- На следующем - аналогично описывают решение подзадач, формулируя уже подзадачи следующего уровня
- Процесс продолжают до тех пор, пока не дойдут до подзадач, алгоритмы решения которых очевидны. При этом, описывая решение каждой задачи, желательно использовать не более одной-двух конструкций, таких как цикл или ветвление

Пример разработки алгоритма методом пошаговой детализации

Словесная постановка задачи

Разработать программу, которая с заданной точностью ε находит значение аргумента *x* по заданному значению функции *y* при известном значении *n*:

$y = \frac{(x+1)^n - 1}{x}$, где $n>1$, $x>0$.

Формальная постановка задачи

Входные данные: eps – точность число, степень $n>1$, y – значение функции

Выходные данные: x – аргумент, $x>0$

Метод решения:

При $n>1$ данная функция является монотонно возрастающей. Для нахождения значения x можно применить метод половинного деления. Суть данного метода заключается в следующем. Вначале определяют отрезок $[x_1, x_2]$ такой, что $f(x_1) \le y \le f(x_2)$. Затем делят его пополам $x_t = (x_1+x_2)/2$ и определяют, в какой половине отрезка находится x, для чего сравнивают $f(x_t)$ и y. Полученный отрезок опять делят пополам и так до тех пор, пока разность x_1 и x_2 не станет меньше заданного значения *eps*.

Для разработки алгоритма программы используем метод пошаговой детализации

Шаг 1. Определяем общую структуру программы

Программа:

Ввести y, n, eps.

Определить x.

Вывести x, y.

Конец.

Шаг 2. Детализируем операцию определения *x*

Определить x:

Определить x1 такое, что f(x1) ≤ y.

Определить x2 такое, что f(x2) ≥ y.

Определить x на интервале [x1,x2].

Все

Шаг 3. Детализируем операцию определения xl. Значение xl должно быть подобрано так, чтобы выполнялось условие $f(xl) \le y$. Известно, что $x > 0$, следовательно, можно взять некоторое значение x, например, xl=l, и последовательно уменьшая его, например в два раза, определить значение xl, удовлетворяющее данному условию.

Определить xl:

```
xl:=l
цикл-пока f(xl)> y
    xl:=xl/2
все-цикл
```

Все

Шаг 4. Детализируем операцию определения $x2$. Значение $x2$ определяем аналогично xl, но исходное значение будем увеличивать в два раза.

Определить x2:

```
x2:=1
цикл-пока f(x2) < y
    x2:=x2*2
все-цикл
```

Все

Шаг 5. Детализируем операцию определения *x*. Определение *x* выполняется последовательным сокращением отрезка [x1,x2].

```
Определить x:
   цикл-пока x2-xl>eps
      Сократить отрезок [xl, x2].
   все-цикл
Все
```

Шаг 6. Детализируем операцию сокращения интервала определения *x*. Сокращение отрезка достигается делением пополам и отбрасыванием половины, не удовлетворяющей условию f $f(x_1) \le y \le f(x_2)$.

```
Сократить интервал определения x:
   xt := (xl + x2)/2
   если f(xt)> y
      то    x2 := xt
      иначе xl := xt
   все-если
Все.
```

Таким образом, за шесть шагов мы разработали весь алгоритм, который выглядит следующим образом

Программа:

```
Ввести y, n, eps
xl:=l
цикл-пока f(xl)> y
    xl:=xl/2
все-цикл
x2:=l
цикл-пока f(x2) < y
    x2:=x2*2
все-цикл
цикл-пока x2-xl>eps
    xt := (xl + x2)/2
    если f(xt)> y
        то    x2 := xt
        иначе xl := xt
    все-если
все-цикл
Вывести xt, y
```

Конец

Теорема Дейкстра

- Алгоритм любой сложности можно реализовать, используя только три конструкции: следования (линейные), выбора (ветвления) и повторения (циклические)
- *Линейным* называется такой вычислительный процесс, при котором все этапы решения задачи выполняются в естественном порядке следования записи этих этапов.
- *Разветвленным* называется такой вычислительный процесс, в котором конкретная последовательность операция зависит от значений одного или нескольких исходных или промежуточных данных (от результатов проверки выполнения какого-либо логического условия)
- *Циклом* называется многократно повторяемый участок вычислений. Вычислительный процесс, содержащий один или несколько циклов, называется циклическим. Циклы делятся на циклы с определенным (заранее заданным) числом повторений (счетные циклы) и циклы с неопределенным числом повторений. Количество повторений последних зависит от соблюдения некоторого условия, задающего необходимость выполнения цикла. Такие циклы носят название итерационных. При этом условие может проверяться в начале цикла – тогда речь идет о цикле с предусловием, или в конце – тогда это цикл с постусловием. Циклические процессы, из которых возможны два варианта выхода: выход по завершении процесса и досрочный выход по какому-либо дополнительному условию – поисковые циклы.

Способы записи алгоритмов

Алгоритм должен быть формализован по некоторым правилам посредством конкретных изобразительных средств

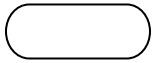
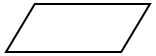
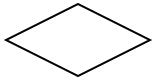
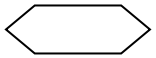
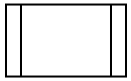
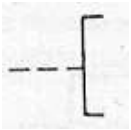
Способы записи алгоритмов:

- словесный или формульно-словесный
- графический (с помощью схем алгоритмов)
- с помощью псевдокода

Блок-схема

- Блок-схемой называется графическое изображение логической структуры алгоритма, в котором каждый этап процесса обработки информации представляется в виде геометрических символов (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций. Перечень символов, их наименование, отображаемые ими функции, форма и размеры определяются ГОСТами.
- Согласно ГОСТ 19.701-90 «СХЕМЫ АЛГОРИТМОВ, ПРОГРАММ, ДАННЫХ И СИСТЕМ. Обозначения условные и правила выполнения» каждой группе действий ставиться в соответствие блок особой формы

ГОСТ 19.701-90: наиболее часто используемые обозначения

Название блока	Обозначение	Назначение блока
Терминатор		Начало или завершение программы
Процесс		Обработка данных: вычисления, пересылки и т.п.
Данные		Операции ввода-вывода данных
Решение		Ветвление, выбор, итерационные и поисковые циклы
Подготовка		Счетные циклы
Предопределенный процесс		Вызов процедур, функций
Соединитель		Маркировка разрывов линий
Комментарий		Пояснения к операциям

Псевдокод

- Псевдокод – описание алгоритма, которое базируется на тех же основных структурах, что и структурные схемы алгоритма

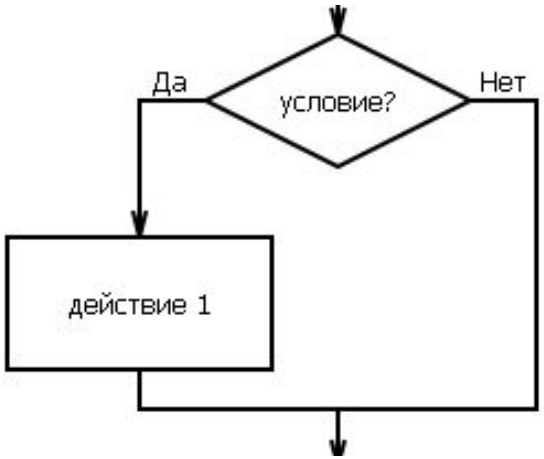
Базовые алгоритмические структуры: следование

<действие1>

<действие 2>

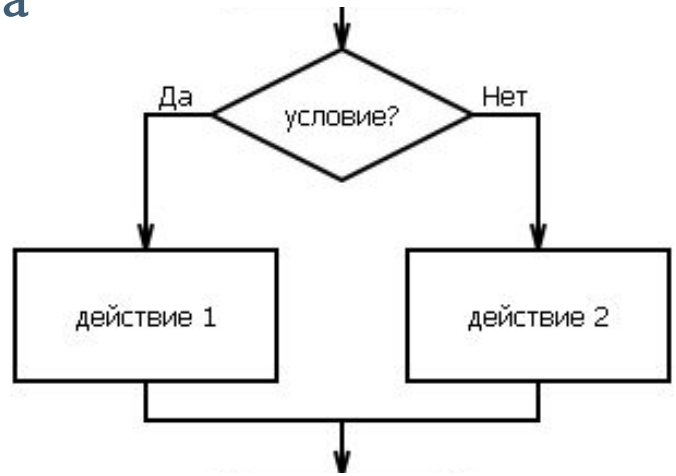
Базовые алгоритмические структуры: ветвление

Обход



Если <условие>
то <действие 1>
Все-если

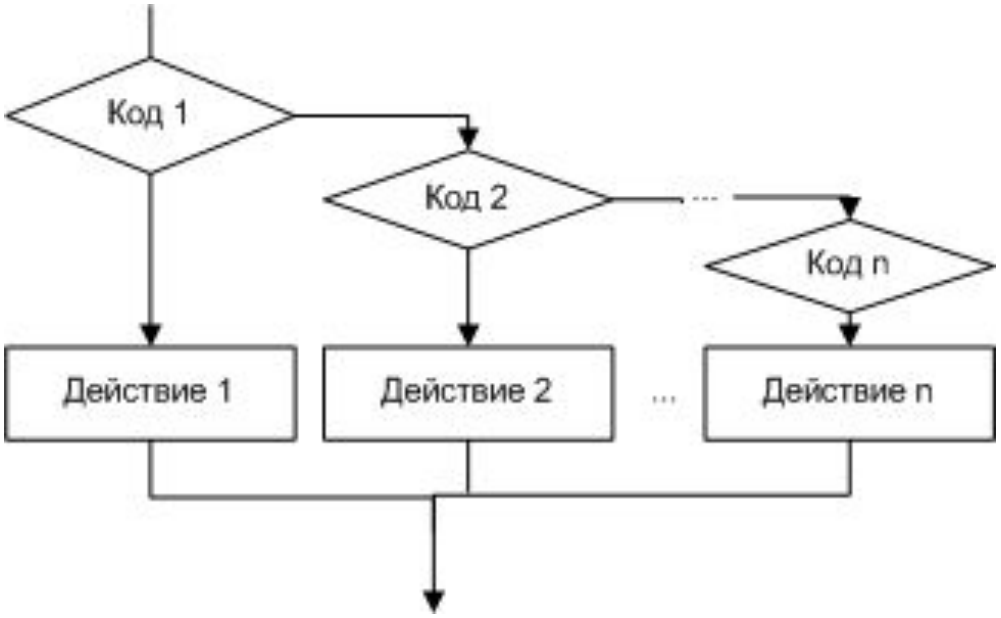
Альтернатива



Если <условие>
то <действие 1>
иначе < действие 2 >
Все-если

Базовые алгоритмические структуры: ветвление

Выбор



Выбор <код>

<код 1>: <действие 1>

<код 2>: <действие 2>

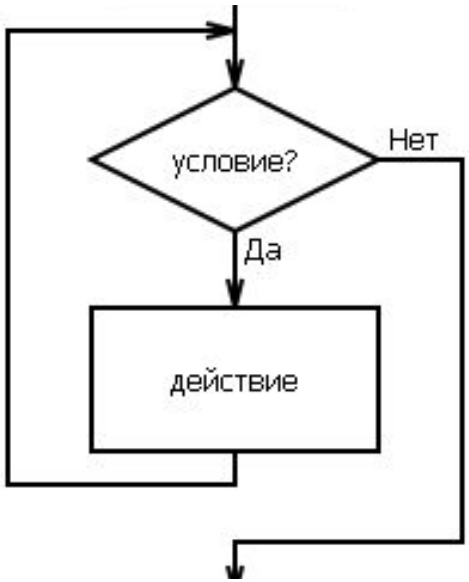
….

<код n>: <действие n>

Все-выбор

Базовые алгоритмические структуры: цикл

С предусловием

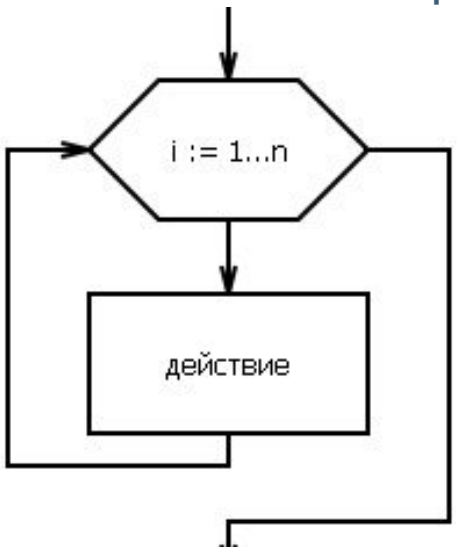


Цикл-пока <условие>
<действие>
Все-цикл

С постусловием

Выполнять
<действие >
пока <условие>
Все-цикл

С известным числом повторений

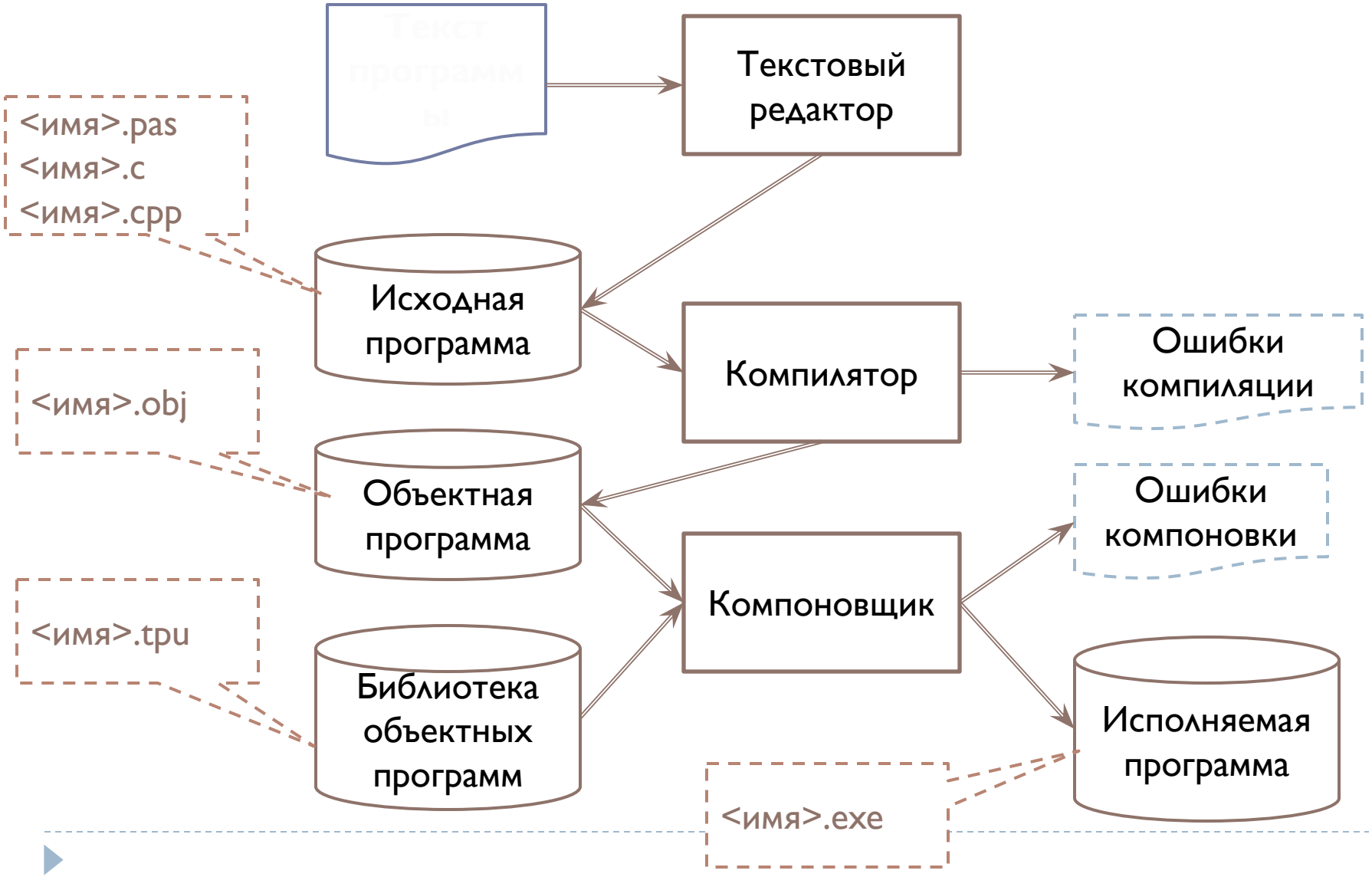


Для <счетчик_цикла>=<n>, <k>, <h>
<действие>
Все-цикл

Реализация

- Реализация – кодирование алгоритма с помощью выбранного языка программирования, тестирование и отладка программы
- *Программа* - запись алгоритма на конкретном формальном языке называется
- Язык может быть определен в техническом задании а может выбираться исходя из особенностей конкретной разработки
- Программа – результат интеллектуального труда, для которого характерно творчество, которое не имеет четких границ. В любой программе присутствует индивидуальность ее разработчика, программа отражает определенную степень искусства программиста. Вместе с тем программирование предполагает и рутинные работы, которые могут и должны иметь строгий регламент выполнения и соответствовать стандартам

Схема процесса подготовки программы к выполнению



Процесс подготовки программы

Основные этапы процесса подготовки программы:

- Создание файла исходного текста и его редактирование при помощи текстового редактора
- Трансляция программы
- Компоновка программы

Текстовый редактор – программа для набора и редактирования исходного текста программы

Компилятор – программа-переводчик исходного текста программы в последовательность машинных команд (машинный код)

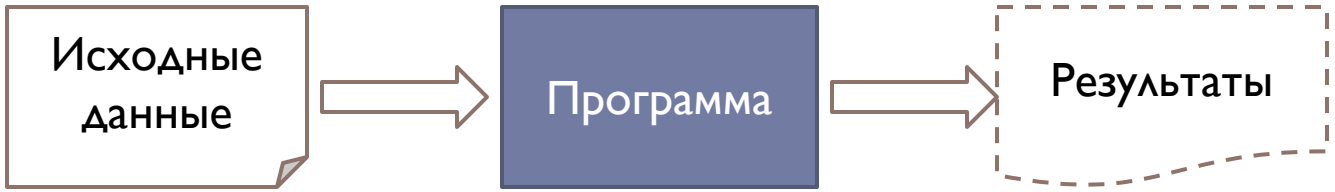
Объектный код – текст программы на машинном языке

Компоновщик – программа сборки отдельных модулей в единую программу

Исполняемая программа – готовая к выполнению программа

Процесс выполнения программы

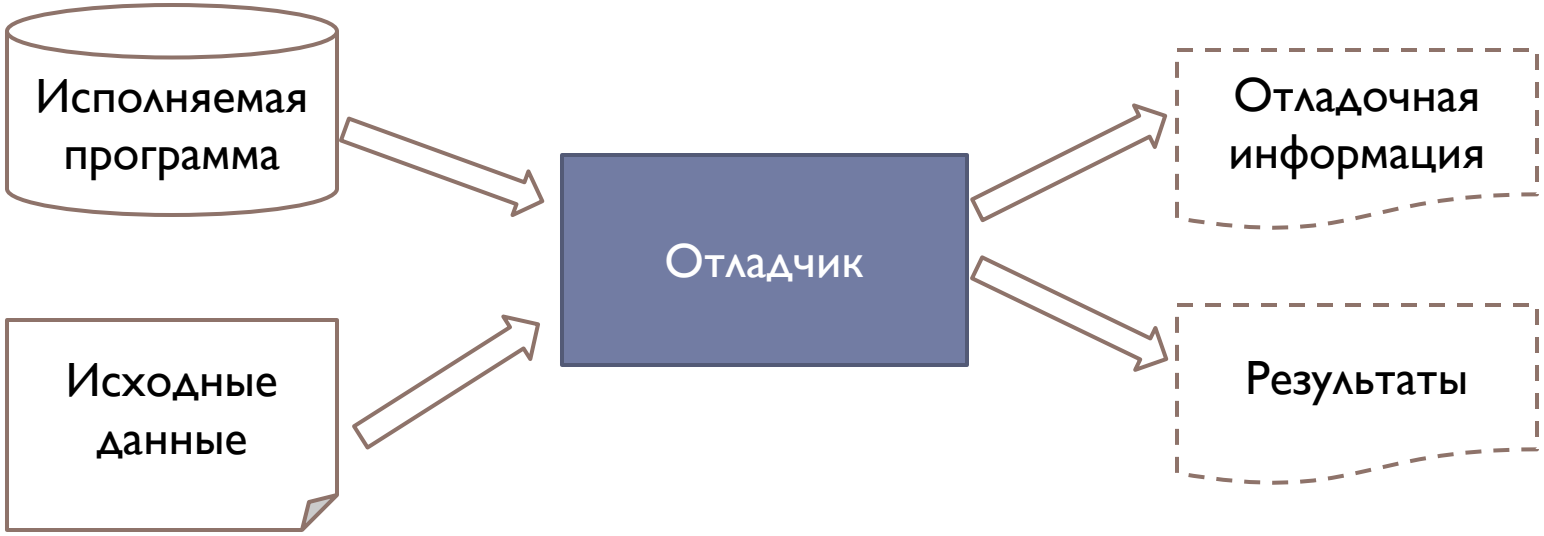
- Ввод исходных данных с клавиатуры | чтение исходных данных из файла
- Обработка данных
- Вывод результатов



- Ошибки выполнения – ситуации, когда продолжение работы программы теряет смысл. Например, «деление на нуль», попытка открыть несуществующий файл
- Для исправления ошибок необходимо их локализовать, т.е. уточнить при выполнении какого фрагмента программы зафиксировано нарушение нормального вычислительного процесса

Отладка программы

- Процесс локализации и исправления ошибок
- Отладчик – специальная программа, которая позволяет выполнить любой фрагмент программы в пошаговом режиме проверить содержимое интересующих нас переменных



Среда языка программирования

- Объединяет специализированный текстовый редактор, компилятор, компоновщик, программу выдачи справочной информации, отладчик и другое программное обеспечение, используемое при разработке программ, в единый пакет

Тестирование

- Процесс выполнения программы при различных тестовых данных с целью обнаружения *логических* ошибок
- Логические ошибки – приводят к выдаче программой неправильных результатов, они не обнаруживаются автоматически компилятором, компоновщиком или операционной системой
- Для поиска логических ошибок необходимо правильно подобрать тестовые данные
- Для поиска логических ошибок можно использовать отладчик

Модификация

В большинстве случаев разработанное ПО через некоторое время требует обновления

Причины выпуска новых версий ПО

- Необходимость исправления ошибок, выявленных в процессе длительной эксплуатации
- Необходимость совершенствования, например, улучшения интерфейса или расширения состава выполняемых функций
- Изменение среды (появление новых технических средств и/или программных продуктов)

Модуль 1.3. Основные принципы программирования

0,5 час

Языки программирования

- **Язык программирования** — формальная знаковая система, предназначенная для записи компьютерных программ
- Язык программирования определяет набор лексических, синтаксических и семантических правил, задающих внешний вид программы и действия, которые выполнит исполнитель (компьютер) под её управлением
- Со времени создания первых программируемых машин человечество придумало более двух с половиной тысяч языков программирования. Каждый год их число увеличивается. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

Понятие системы программирования

Система программирования — это система для разработки новых программ на конкретном языке программирования

В системы программирования входят:

- компилятор или интерпретатор
- интегрированная среда разработки
- средство создания и редактирования текста программы
- обширные библиотеки стандартных программ и функций
- отладочные программы
- справочная система

Транслятор (англ. translator - переводчик) — это программа-переводчик. Она преобразует программу, написанную на одном из языков высокого уровня в программу, состоящую из машинных кодов.

Компилятор (англ. compiler - составитель) читает всю программу целиком, делает её перевод и создаёт законченный вариант программы на машинном языке, который затем и выполняется.

Интерпретатор (англ. interpreter - истолкователь) — переводит и выполняет программу строку за строкой.

Современные интегрированные среды программирования

Интегрированная среда разработки, ИСР (англ. *IDE, Integrated development environment или integrated debugging environment*) — система программных средств, используемая программистами для разработки программного обеспечения

Обычно среда разработки включает в себя: текстовый редактор; компилятор и/или интерпретатор; средства автоматизации сборки; отладчик

Иногда содержит также средства для интеграции с системами управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя. Многие современные среды разработки также включают *браузер классов*, *инспектор объектов* и *диаграмму иерархии классов* — для использования при объектно-ориентированной разработке ПО. Хотя и существуют ИСР, предназначенные для нескольких языков программирования — такие, как Eclipse, NetBeans, Embarcadero RAD Studio, Qt Creator или Microsoft Visual Studio, но обычно ИСР предназначается для одного определённого языка программирования - как, например, Visual Basic,PureBasic, Delphi, Dev-C++

Методологии программирования

- **Структурное программирование** — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 70-х годах XX века Э. Дейкстрой, разработана и дополнена Н. Виртом
- **Объектно-ориентированное**, или **объектное**, программирование (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов

Принципы структурного программирования

- Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:
  - **Следование**
  - **Ветвление**
  - **Цикл**
- В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.
- Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться в виде подпрограмм - процедур или функций. В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция **вызова подпрограммы**. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы.
- Разработка программы ведётся пошагово, методом «сверху вниз».