

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра веб-технологий и компьютерного моделирования**

А. С. Кравчук, А. И. Кравчук, Е. В. Кремень

**ВВЕДЕНИЕ
В АЛГОРИТМИЗАЦИЮ
И ПРОГРАММИРОВАНИЕ**

**Учебные материалы
для студентов специальности 6-05-0533-07
«Математика и компьютерные науки
(по профилизациям)»**

**МИНСК
2023**

УДК 004.421(075.8)
ББК 32.973-018я73-1
К78

Рекомендовано советом
механико-математического факультета БГУ
30 мая 2023 г., протокол № 9

Рецензент
профессор кафедры экономической информатики
Белорусского государственного экономического университета
кандидат технических наук, доцент *А. М. Седун*

Кравчук, А. С.
К78 Введение в алгоритмизацию и программирование : учеб.
материалы для студентов специальности 6-05-0533-07 «Математика
и компьютерные науки (по профилизациям)» / А. С. Кравчук,
А. И. Кравчук, Е. В. Кремень. – Минск : БГУ, 2023. – 23 с.

Изложены основные свойства алгоритма, типы алгоритмических процессов, средства описания алгоритма, дается понятие как структурного (императивного), так и объектно ориентированного программирования, рассматривается история разработки языков программирования, а также вопросы, связанные с проектированием программного обеспечения. Издание предназначено для студентов, начинающих изучать программирование.

УДК 004.421(075.8)
ББК 32.973-018я73-1

© Кравчук А. С., Кравчук А. И.,
Кремень Е. В., 2023
© БГУ, 2023

Оглавление

Введение.....	4
Этапы решения задач в программировании. Алгоритмизация	5
Свойства алгоритма	6
Типы алгоритмических процессов.....	6
Средства описания алгоритма	7
Графический способ описания алгоритма	8
Другие способы описания алгоритмов.....	14
Программирование	16
Понятие структурного (императивного) программирования	16
Языки низкого уровня	17
История языков высокого уровня для структурного программирования.....	18
Объектно-ориентированное программирование (ООП).....	19
Стандарты языка C++	19
Проектирование программного обеспечения	20
Этапы разработка программного обеспечения согласно ГОСТ	21
Литература	23

Введение

Целью дисциплины является формирование, развитие и становление у студента следующих основных общекультурных и профессиональных компетенций:

- способность к анализу информации и/или сущности задачи;
- постановке формальной цели и выбору путей ее достижения;
- умение самостоятельно разработать алгоритм решения задачи.

На самом деле алгоритмизация и программирование являются неотъемлемой частью нашего мира, но большая часть людей даже не подозревает об этом.

Формально понятия алгоритмизации и программирования существенно отличаются друг от друга, хотя на практике образуют симбиоз.

Этапы решения задач в программировании. Алгоритмизация

Можно выделить следующие основные этапы решения задачи:

- постановка (формулировка) задачи;
- построение модели, выбор метода решения задачи;
- разработка алгоритма;
- проверка правильности алгоритма;
- анализ алгоритма и его сложности;
- реализация алгоритма;
- отладка программы, обнаружение, локализация и устранение возможных ошибок;
- составление документации.

Прежде чем понять задачу, необходимо ее четко *сформулировать*. Это условие не является достаточным для понимания задачи, но оно абсолютно необходимо.

Следующий важный шаг в решении задачи – формулировка для нее *математической модели*. Выбор модели и реализация алгоритма в значительной степени может повлиять на эффективность алгоритма решения задачи. Все перечисленные выше этапы нельзя рассматривать независимо друг от друга. В особенности первые три сильно влияют на последующие.

Понятие алгоритма занимает центральное место в вычислительной математике и программировании. Справедливо следующее определение. . *Алгоритм* – это конечная совокупность точно заданных действий, для решения некоторой задачи.

Алгоритм означает точное описание некоторого процесса (инструкцию по его выполнению). Разработка алгоритма является сложным и трудоемким процессом. *Алгоритмизация* – это техника разработки (составления) алгоритма (плана) для решения, поставленной задачи *не только на компьютере*.

Наиболее распространенная процедура *доказательства правильности алгоритма* – это прогон ее на ранних тестах. Однако это не исключает все сомнения, т.к. написание программы также осуществляется людьми – главным источником ошибок.

Задача анализа алгоритма и его сложности – получение оценок или границ для объема памяти или времени работы алгоритма. Полный анализ способен выявить узкие места в программах.

Реализация алгоритма – процесс корректного преобразования алгоритма в программу на одном из языков программирования. Требует

также построения целой системы структур данных для представления модели.

Эксплуатации программы предшествует *отладка* – исправление синтаксических и логических ошибок. Процесс проверки программы включает экспериментальное подтверждение того факта, что программа делает именно то, что должна делать. Обычно множество вариантов входных данных огромно, и полная проверка невозможна. Необходимо выбрать множество вводов, которые проверяют каждый участок программы. Программы следует тестировать также для того, чтобы определить их вычислительные ограничения.

Этап *документирования* не является последним шагом в процессе построения алгоритма. Он должен переплетаться со всем процессом построения алгоритма для того, чтобы была возможность понять программные модули, написанные другими.

Свойства алгоритма

Характерными свойствами алгоритма являются определенность, массовость и результативность.

Определенность (детерминированность) алгоритма предполагает такое составление предписания, которое не оставляет места для различных толкований или искажений результата, т.е. последовательность действий алгоритма строго и точно определена.

Массовость определяет возможность использования любых исходных данных из некоторого допустимого множества. Правило, сформулированное только для данного случая, не является алгоритмом (например, таблица умножения не является алгоритмом, а правило умножения «столбиком» уже алгоритм).

Результативность (конечность) алгоритма означает, что при любом допустимом исходном наборе данных алгоритм закончит свою работу за конечное число шагов.

Типы алгоритмических процессов

По структуре выполнения алгоритмы и программы делятся на три вида:

- линейные;
- разветвляющиеся;
- циклические.

Линейный алгоритм – это такой алгоритм, в котором все действия выполняются последовательно друг за другом и только один раз. Схема представляет собой линейную последовательность блоков, которые

располагаются сверху вниз в порядке их выполнения. Первичные и промежуточные данные *не* оказывают влияния на *изменение* направления процесса вычисления, т.е. *линейный* алгоритм не содержит логических условий, имеет одну ветвь обработки.

Разветвляющийся алгоритм – это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.

Для решения многих задач характерно многократное повторение отдельных участков вычислений. Для решения таких задач применяются *циклические алгоритмы*. *Цикл* – последовательность команд, которая повторяется до тех пор, пока будет выполнено заданное условие. Циклический алгоритм содержит один или несколько циклов. Цикл, *не* содержащий внутри себя других циклов, называют *простым*. Если он содержит внутри себя другие циклы, то цикл называют *сложным* или *вложенным*. Любой цикл характеризуется одной или несколькими переменными, называемыми *параметрами цикла*, от анализа значений которых зависит выполнение цикла. *Параметр цикла* – переменная, принимающая при каждом вхождении в цикл новое значение. Условное

Средства описания алгоритма

Для записи алгоритма решения задачи применяются следующие изобразительные способы их представления:

- блок-схема (схема графических символов);
- словесно-формульное описание;
- алгоритмические языки;
- операторные схемы;
- псевдокод.

Для записи алгоритма существуют *общие требования*:

- каждый алгоритм должен иметь имя, которое раскрывает его смысл;
- необходимо обозначить начало и конец алгоритма;
- описать входные и выходные данные;
- указать команды, которые позволяют выполнять определенные действия над выделенными данными.

Отсюда следует *общий вид алгоритма*:

- название алгоритма;
- описание данных;
- начало;
- команды;
- конец.

Графический способ описания алгоритма

Графический способ описания алгоритма (*блок - схема*) получил самое широкое распространение. Для графического описания алгоритмов используются схемы алгоритмов или блочные символы (блоки), которые соединяются между собой линиями связи.

Каждый этап вычислительного процесса представляется геометрическими фигурами (блоками). Они делятся на арифметические или вычислительные (прямоугольник), логические (ромб) и блоки ввода-вывода данных (параллелограмм) и др. (Рисунок 1).





	<i>Начало и конец алгоритма (для функций «Вход», «Выход»)</i>
	<i>Блок обработки. Внутри блока записываются формулы, обозначения и функции</i>
	<i>Блок условия. Внутри блока записываются условия выбора направления действия алгоритма</i>
	<i>Блок предопределенного процесса (функция/подпрограмма)</i>
	<i>Блок ввода информации</i>
	<i>Блок цикла с известным количеством повторений</i>
	<i>Блок вывода информации на печатающее устройство</i>
	<i>Соединительные блоки (на этой же или следующей странице)</i>

Рисунок 1 - Условные обозначения, используемые в блок-схемах

Порядок выполнения этапов обычно указывается стрелками, соединяющими блоки. Геометрические фигуры размещаются сверху вниз и слева на право. Нумерация блоков производится в порядке их размещения в схеме.

Замечание.

Блоки «Начало» и «Конец» в блок-схеме **не** нумеруются.

Знак « \leftarrow » означает: внести значение переменной в ячейку памяти с определенным именем (*операция присваивания*).

Пример блок-схемы линейного вычислительного процесса

Составим блок-схему вычисления $y = f(x), z = \chi(y)$, где $f(), \chi()$ – известные функции, при заданном значении переменной x (Рисунок 2).

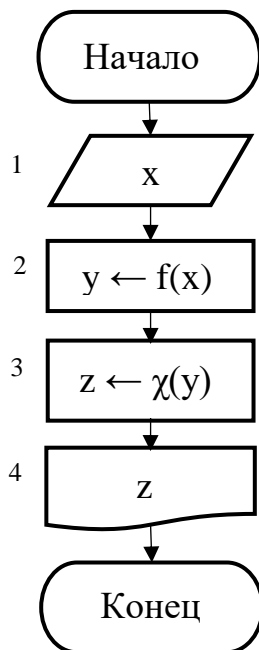


Рисунок 2 - Блок-схема примера линейного алгоритма

Пример блок-схемы разветвляющегося алгоритма

Условное изображение обязательного фрагмента разветвления алгоритма представлено на рисунке ниже (Рисунок 3).

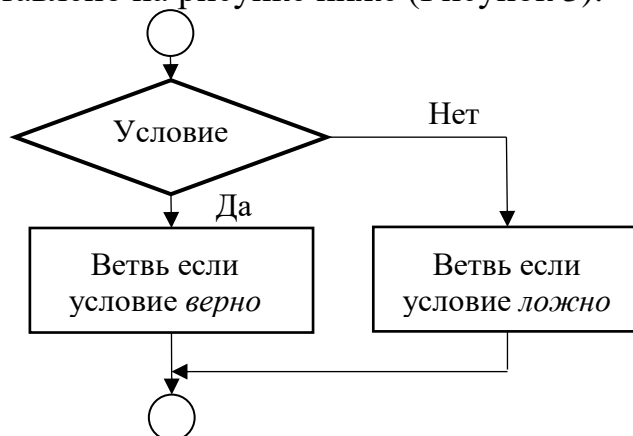


Рисунок 3 - Условное изображение фрагмента разветвляющегося алгоритма

Определим попадает ли точка внутрь круга, если радиус круга известен, а его центр и точка задаются введенными с консоли координатами (Рисунок 4).

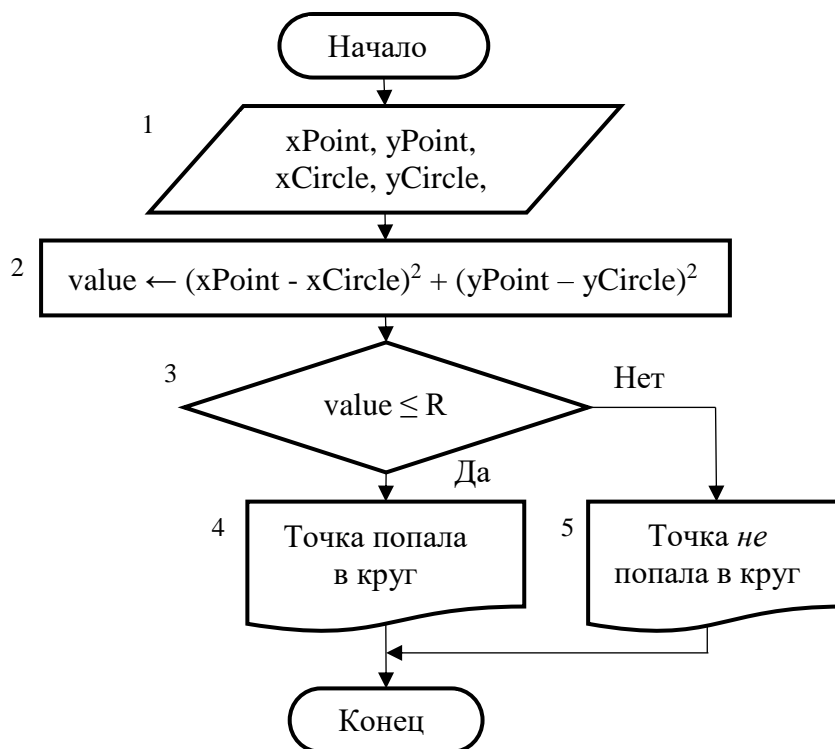


Рисунок 4 - Блок-схема примера разветвляющегося алгоритма

Построение схем циклических вычислительных процессов

Условное изображение обязательного фрагмента циклического алгоритма представлено на рисунке ниже (Рисунок 5).

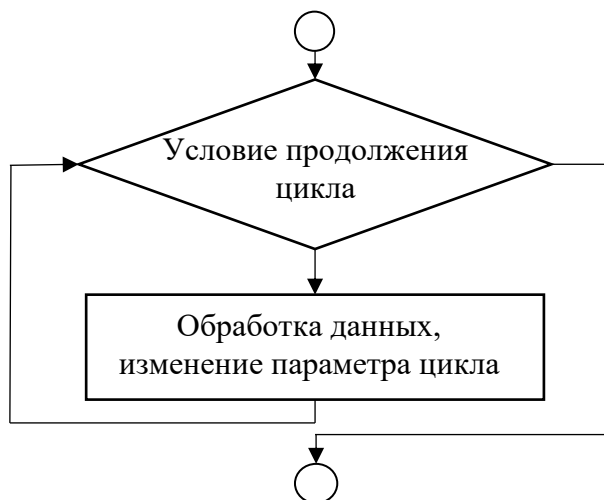


Рисунок 5 - Условное изображение фрагмента циклического алгоритма

Существуют циклы с *известным числом повторений* (параметром цикла является целое число) и итерационные циклы (в большинстве случаев параметром цикла является число с плавающей точкой). При итерационном цикле выход из тела цикла, как правило, происходит при достижении заданной точности вычисления.

Циклическое описание многократно повторяемых процессов значительно снижает трудоемкость написания программ.

Пример.

Составить блок-схему нахождения суммы n первых целых чисел от 0 до $n-1$ (Рисунок 6, Рисунок 7).

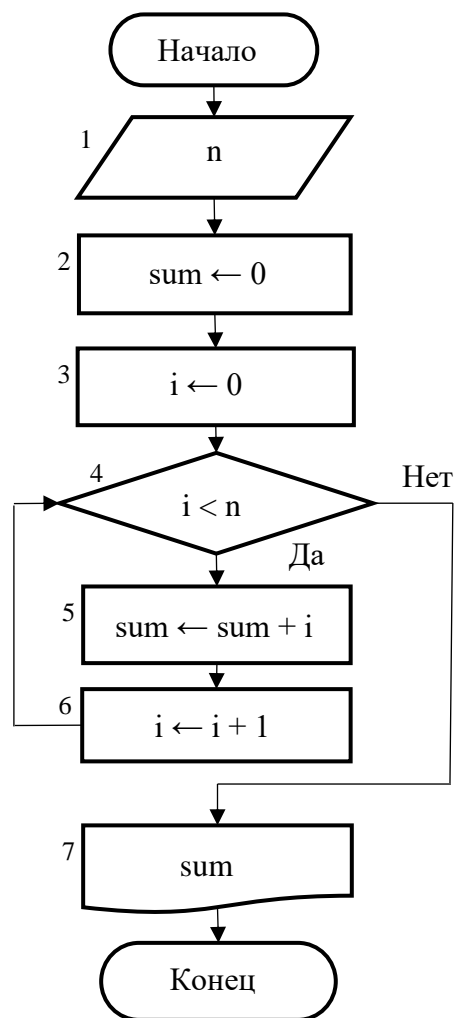


Рисунок 6 - Обобщенная (универсальная) блок-схема циклического алгоритма

Блок-схема любого конечного *циклического* процесса (Рисунок 6) независимо от многообразия сводящихся к нему задач должна содержать блок задания начального значения параметра цикла (блок 3), блок проверки

условия продолжения цикла (блок 4), блок реализации необходимых вычислений (блок 5) и блок изменения параметра цикла (блок 6).

Способ организации цикла зависит от условия задачи. Во многих задачах количество повторений цикла известно (как в примере 1). Это так называемые *циклы с известным количеством повторений* или *циклы со счетчиком* (целочисленным параметром). В отличие от общей схемы циклического процесса, приведенной на рисунке выше (Рисунок 6) они изображаются несколько в другом виде, позволяющем в этом случае существенно сократить блок-схему (Рисунок 7).

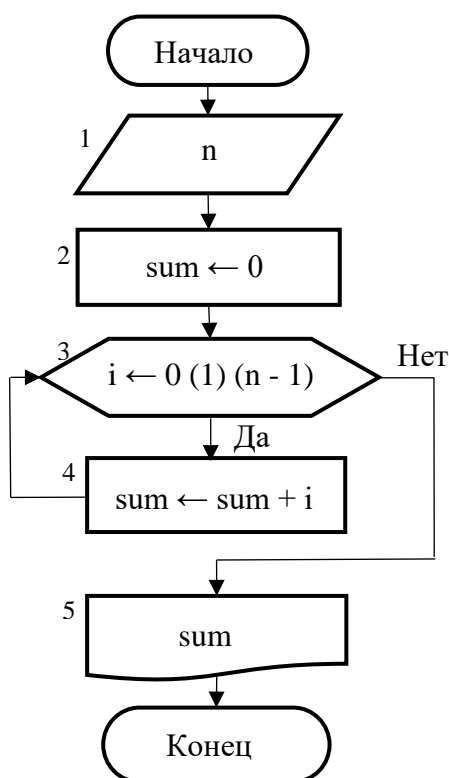


Рисунок 7 - Блок схема циклического алгоритма с известным числом повторений

Варианты заданий

По линейным алгоритмам

1. Даны x, y . Получить $\frac{|x| - |y|}{1 + |xy|}$.
2. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.
3. Вычислить расстояние между двумя точками с данными координатами.

4. Найти площадь треугольника со сторонами a, b, c по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \text{ где } p = \frac{(a+b+c)}{2}.$$

5. Даны x, y, z . Найти w , если $w = x - \frac{x^2}{1 + \sin^2(x+y+z)}$;

6. Даны x, y, z . Найти w , если $w = \cos\left(z^2 + \frac{x^2}{4} + y\right)$;

По разветвляющимся алгоритмам

1. Даны два числа. Вывести первое число, если оно больше второго, и оба числа, если это не так.

2. Найти наименьшее из трех данных чисел a, b, c .

3. Даны три числа a, b, c , являющиеся длинами сторон треугольника. Определить тип треугольника (равносторонний, равнобедренный, разносторонний).

4. Даны x, y, z . Найти: $\max\{x+y+z, xyz\} + 3$;

5. Даны x, y, z . Найти: $\min\{x^2 + y^2, y^2 + z^2\} - 4$.

6. Дано x . Вычислить y , если: $y = \begin{cases} x^2, & \text{при } 2 \leq x \leq 2, \\ 4, & \text{при } x < -2 \text{ или } > 2 \end{cases}$;

7. Дано x . Вычислить y , если: $y = \begin{cases} 0, & \text{при } x \leq 0 \\ x, & \text{при } 0 < x \leq 1, \\ x^4, & \text{при } x > 1 \end{cases}$

8. Дано x . Вычислить y , если: $y = \begin{cases} x^2 + 4x + 5, & \text{при } x \leq 2, \\ \frac{1}{x^2 + 4x + 5}, & \text{при } x > 2. \end{cases}$

9. Дано x . Вычислить y , если: $y = \begin{cases} 0, & \text{при } x \leq 0, \\ x^2 - x, & \text{при } 0 < x \leq 1, \\ x^2 - \sin(\pi \cdot x^2) - 1, & \text{при } x > 1 \end{cases}$.

По циклическим алгоритмам

Дано натуральное n . Вычислить значение суммы или произведения.

1. $\frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n}$;

2. 2^n ;

3. $\left(1 + \frac{1}{1^2}\right) \left(1 + \frac{1}{2^2}\right) \dots \left(1 + \frac{1}{n^2}\right)$

4. $\frac{1}{1^5} + \frac{1}{2^5} + \dots + \frac{1}{n^5}$

5. $\frac{1}{3^2} + \frac{1}{5^2} + \dots + \frac{1}{(2n+1)^2}$

6. $\frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + \dots + \frac{1}{\sin 1 + \sin 2 + \dots + \sin n}$

Другие способы описания алгоритмов

Формульно-словесный способ записи алгоритма характеризуется тем, что описание осуществляется с помощью слов и формул. Содержание последовательности этапов выполнения алгоритмов записывается на естественном профессиональном языке предметной области в произвольной форме.

Алгоритмические языки - это формальные языки, специально разработанные для записи алгоритмов. Причем для каждого алгоритмического языка существуют свои особенности, это:

- *алфавит* – или набор используемых символов;
- *синтаксис* – система правил, по которым из алфавита образуются правильные конструкции языка;
- *семантика* – система правил, строго определяющая смысл и способ употребления каждой конструкции языка.

Существует большое число алгоритмических языков в том числе на основе кириллицы.

Операторные схемы алгоритмов. Суть этого способа описания алгоритма заключается в том, что каждый оператор обозначается буквой (например, А – арифметический оператор, Р – логический оператор и т.д.).

Операторы записываются слева направо в последовательности их выполнения, причем, каждый оператор имеет индекс, указывающий порядковый номер оператора. Алгоритм записывается в одну строку в виде последовательности операторов.

Псевдокод – система команд абстрактной машины (Рисунок 8, Рисунок 9). Этот способ записи алгоритма с помощью операторов близких к алгоритмическим языкам.

```

    Алгоритм 1: Как прочитать книгу
    Исходные параметры: experiment.data
    Результат: output, xoptimal
    x=0; до тех пор, пока  $\tau_{\text{norm}} > \epsilon_{\text{tol}}$  выполнять
    |  $s_{k-1} \leftarrow x_k - x_{k-1}$ ; // Step length computation: если  $k$  is even тогда
    |    $\alpha_k^{ABB} = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$ 
    |   иначе
    |      $\alpha_k^{ABB} = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$ 
    |   конец условия
    |  $k \leftarrow k + 1$ ; цикл  $i = l$  выполнять
    |    $x_{i+1} = P_{\Omega}(x_i - \alpha_k^{ABB} * g_k)$ ;
    |   конец цикла
    // Compute the termination constant  $\tau_{\text{norm}} = \text{abs}(\|x_k\|_2 - \|x_{k-1}\|_2)$ 
    конец цикла
  
```

Рисунок 8 – Пример псевдокода

Dijkstra(G, l, s)

- ▷ Вход: граф $G(V, E)$ (ориентированный или нет) с неотрицательными
- ▷ длинами рёбер $\{l_e : e \in E\}$; вершина $s \in V$.
- ▷ Выход: для всех вершин u , достижимых из s ,
- ▷ $dist[u]$ будет равно расстоянию от s до u (и ∞ для недостижимых).

for всех вершин $u \in V$

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow \text{NIL}$

$dist[s] \leftarrow 0$

$H \leftarrow \text{MAKE-QUEUE}(V)$ (в качестве ключей используются значения $dist$)

while $H \neq \emptyset$

$u \leftarrow \text{DELETE-MIN}(H)$

for всех рёбер $(u, v) \in E$

if $dist[v] > dist[u] + l(u, v)$

then $dist[v] \leftarrow dist[u] + l(u, v)$

$prev[v] \leftarrow u$

$\text{DECREASE-KEY}(H, v, dist[v])$

Рисунок 9 - Алгоритм Дейкстры для нахождения кратчайших путей

Программирование

Программирование - раздел информатики, область знаний об алгоритмах и программах, их свойствах, а также исполнителях алгоритмов и программ.

В узком смысле *программирование* – процесс создания программного обеспечения, которое включает в себя:

- анализ;
- проектирование (при использовании процедурного программирования этот раздел является алгоритмизацией в узком смысле этого слова);
- кодирование и компиляцию;
- тестирование и отладку.

В еще более узком смысле слова *программирование* рассматривается как кодирование алгоритмов на заданном языке программирования. При этом под программированием также может пониматься разработка логической схемы для программируемых логических интегральных схем (ПЛИС), а также процесс записи информации в постоянном запоминающем устройстве.

Понятие структурного (императивного) программирования

В 70-е годы прошлого века возник новый подход к разработке алгоритмов и программ, который получил название структурного проектирования программ. К его достоинствам можно отнести: более высокую производительность; читаемость программ; простоту тестирования и эффективность программ.

Структурное программирование — парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков.

В соответствии с парадигмой, все операции в программе, построенной на основе структурного программирования, должны представлять собой либо исполняемые в линейном порядке выражения, либо одну из следующих управляющих конструкций: вызовы подпрограмм; вложенные на произвольную глубину операторы условия; циклические операторы.

Языки низкого уровня

Первые программы писались на машинном языке (в то время компьютер назывался вычислительной машиной), т.к. для компьютеров того времени еще не существовало развитого программного обеспечения, а машинный язык — это единственный способ взаимодействия с аппаратным обеспечением компьютера.

Каждую команду машинного языка напрямую выполняет то или иное электронное устройство.

Данные и команды записывали в цифровом виде (например, в шестнадцатеричной или двоичной системах счисления).

Понять программу на таком языке очень сложно; кроме того, даже небольшая программа получалась состоящей из множества строк кода. Ситуация усложнялась еще и тем, что каждый тип вычислительной машины в зависимости от архитектуры процессора понимал лишь свой машинный язык.

Людам, в отличие от машин, более понятны слова, чем наборы цифр. Стремление человека оперировать словами, а не цифрами привело к появлению *ассемблеров*. Это языки, в которых вместо численного обозначения команд и областей памяти используются словесно-буквенные.

При этом появляется проблема: машина не в состоянии понимать слова. Необходим какой-нибудь переводчик на ее родной машинный язык. Поэтому, начиная со времен ассемблеров, под каждый язык программирования создаются *трансляторы* — специальные программы, преобразующие программный код с языка программирования в машинный код.

Ассемблеры на сегодняшний день продолжают использовать. В системном программировании с их помощью создаются низкоуровневые интерфейсы операционных систем, компоненты драйверов.

Написание программ на ассемблере является довольно сложной задачей, к тому же необходимы знания устройств компьютера. И тем не менее программы на ассемблере — самые эффективные и работоспособные.

После *ассемблеров* наступил рассвет языков так называемого высокого уровня. Для этих языков потребовалось разрабатывать более сложные трансляторы, т. к. языки высокого уровня куда больше удобны для человека, чем для вычислительной машины.

В отличие от *ассемблеров*, которые остаются привязанными к своим типам вычислительных процессоров, языки высокого уровня обладают переносимостью. Это значит, что, написав один раз программу на языке программирования высокого уровня, программист может выполнить ее на любом компьютере.

История языков высокого уровня для структурного программирования

В 1954 г. была начата разработка первого компилятора языка высокого уровня. Через два года появился язык FORTRAN (FORmula TRANslator – «переводчик формул»). Язык содержал средства, которые значительно упрощали разработку, однако программирование на FORTRAN не было простой задачей: если в коротких программах код был легко понимаемым, то, когда дело касалось больших программ их текст становился нечитаемым. Несмотря на это, язык был довольно успешным и было выпущено много его версий.

Проблемы были решены после разработки других языков структурного программирования: в них появилась возможность создания программных блоков, независимых подпрограмм, поддержки рекурсии и локальных переменных, отсутствие оператора безусловного перехода (GoTo).

Таким образом, такие языки стали поддерживать возможность разбиения программы на составляющие элементы. На протяжении десяти лет было создано достаточно большое число новых языков: ALGOL (1958 г.) предназначался для записи алгоритмов, которые составлены из обособленных блоков; COBOL (1959 г.) использовался для массовой обработки данных в сферах управления и бизнеса; BASIC (1965 г.) позволял писать простые программы, использовался для обучения основам программирования.

Особого внимания заслуживает язык PASCAL (1970 г.), названный в честь ученого Блеза Паскаля, который использовался как для обучения, так и для решения задач различной сложности. Программы на PASCAL легко читаемы, что позволяет быстро находить и исправлять ошибки, также он хорошо структурирован. Все вышеперечисленное привело к его широкой распространенности, и даже в данное время его активно используют в учебных заведениях.

В 1972 г. появился язык C, что стало успешным шагом в программировании. Язык сочетал в себе преимущества многих языков и обладал большим числом разных нововведений. Широкие возможности, структурированность и относительная простота его изучения позволили языку быстро стать признанным и завоевать место одного из основных языков. Появление структурного программирования дало отличные результаты, но все же было еще сложно писать длинные и серьезные программы.

Объектно-ориентированное программирование (ООП)

С 1970-х гг. были заложены основы объектно-ориентированного программирования (ООП), которое возникло как продолжение развития процедурного программирования, при котором данные и подпрограммы их обработки формально не были связаны. ООП включает следующие основные понятия:

- класс;
- объект;
- инкапсуляция;
- наследование;
- полиморфизм.

В настоящее время число прикладных языков программирования, которые реализуют объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам. Основные языки, которые поддерживают концепцию ООП: C++, C#, Object Pascal (Delphi), Java и т.д.

Стандарты языка C++

Важную роль в развитии языка C++ играют стандарты языка. Хотя язык C++ разрабатывался с 1980-х годов, первый стандарт языка C++98 был окончательно утвержден только в 1998 году.

В 2003 году был издан стандарт C++03, являющийся уточнением стандарта C++98.

Наиболее существенные изменения языка произошли в стандарте C++11, разработка которого завершилась в 2011 году.

В 2014 году был издан стандарт C++14, не содержащий существенных изменений, а только устраняющий ряд дефектов стандарта C++11.

В 2017 году был издан стандарт C++17.

В 2020 году разработан и утвержден стандарт C++20, который существенно изменил сам язык.

В феврале 2023 года комитет ISO по C++ закончил техническую работу над спецификацией языка программирования C++ 23, объявив ее функционально завершенной, и готовит финальный документ для утверждения.

Проектирование программного обеспечения

Проектирование программного обеспечения - процесс создания проекта программного обеспечения (ПО), а также дисциплина, изучающая методы проектирования.

Целью проектирования является определение внутренних свойств системы и детализации ее внешних (видимых) свойств на основе выданных заказчиком требований к ПО (исходные условия задачи). Эти требования подвергаются анализу.

Первоначально программа рассматривается как черный ящик. Ход процесса проектирования и его результаты зависят не только от состава требований, но и выбранной модели процесса, опыта проектировщика.

Модель предметной области накладывает ограничения на бизнес-логику и структуры данных. В зависимости от класса, создаваемого ПО, процесс проектирования может обеспечиваться как «ручным» проектированием, так и различными средствами его автоматизации.

В процессе проектирования ПО для выражения его характеристик используются различные нотации — блок-схемы, UML-диаграммы и др. средства.

Проектированию обычно подлежат:

- архитектура ПО;
- устройство компонентов ПО;
- пользовательские интерфейсы.

Проектирование ведется *поэтапно* в соответствии со стадиями:

- техническое задание;
- техническое предложение;
- эскизный проект;
- технический проект;
- рабочий проект.

На каждом из этапов формируется свой комплект документов, называемый проектом (проектной документацией).

Жизненный цикл программного обеспечения (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент полного прекращения его поддержки.

Этапы разработка программного обеспечения согласно ГОСТ

Может показаться, что стадии и этапы разработки программных продуктов полностью зависят от доброй воли и желания программиста, но это не так. Редко создание программного продукта происходит силами одного программиста изолированно от заказчика, потребителя, специалистов и т. д. Как правило такие случаи возможны только во время обучения языку. Республика Беларусь входит в состав Евразийского экономического союза, члены которого в целях унификации, применяемых при создании, эксплуатации и развитии интегрированной информационной системы применяют общие стандарты и рекомендации в области информационно-телекоммуникационных технологий и информационной безопасности. В частности, недавно введен в обращение российский стандарт ГОСТ Р 59793–2021 «Информационные технологии. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания».

Этот стандарт предусматривает следующие *стадии и этапы создания автоматизированной системы (АС)*:

1. формирование требований к АС:
 - a. обследование объекта и обоснование необходимости создания АС;
 - b. формирование требований пользователя к АС;
 - c. оформление отчета о выполненной работе;
2. разработка концепции АС;
 - a. изучение объекта;
 - b. проведение необходимых научно-исследовательских работ;
 - c. разработка вариантов концепции АС и выбор варианта концепции АС, удовлетворяющего требованиям пользователей;
 - d. оформление отчета о проделанной работе;
3. техническое задание:
 - a. разработка и утверждение технического задания на создание АС;
4. эскизный проект:
 - a. разработка предварительных проектных решений по системе и ее частям;
 - b. разработка документации на АС и ее части;
5. технический проект:
 - a. разработка проектных решений по системе и ее частям;

- b. разработка документации на АС и ее части;
 - c. разработка и оформление документации на поставку изделий для комплектования АС и (или) технических требований (технических заданий) на их разработку;
 - d. разработка заданий на проектирование в смежных частях проекта объекта автоматизации;
6. рабочая документация:
- a. разработка рабочей документации на АС и ее части;
 - b. разработка и адаптация отдельных видов обеспечения АС;
7. ввод в действие:
- a. подготовка объекта автоматизации к вводу АС в действие;
 - b. подготовка персонала;
 - c. комплектация АС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями);
 - d. строительные-монтажные работы;
 - e. пусконаладочные работы;
 - f. проведение предварительных испытаний;
 - g. проведение опытной эксплуатации;
 - h. проведение приемочных испытаний;
8. сопровождение АС:
- a. выполнение работ в соответствии с гарантийными обязательствами;
 - b. послегарантийное обслуживание.

Эскизный, технический проекты и рабочая документация — это последовательное построение все более точных проектных решений. Допускается исключать стадию «Эскизный проект» и отдельные этапы работ на всех стадиях, объединять стадии «Технический проект» и «Рабочая документация» в «Технорабочий проект», параллельно выполнять различные этапы и работы, включать дополнительные.

Литература

1. Пацей, Н.В. Основы алгоритмизации и программирования / Н.В. Пацей – Минск : БГТУ, 2010. – 289 с.
2. Кравчук, А.И. Сборник лабораторных работ и примеров решения задач по алгоритмизации и программированию на языке Си / А.И. Кравчук, А.С. Кравчук – Минск: Технопринт, 2002. – 116 с.
3. ГОСТ Р 59793–2021 «Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания - [Электронный документ] URL: <https://protect.gost.ru/document.aspx?control=7&id=241692> (дата обращения 01.08.22)

Учебное издание

Кравчук Александр Степанович
Кравчук Анжелика Ивановна
Кремень Елена Васильевна

ВВЕДЕНИЕ
В АЛГОРИТМИЗАЦИЮ
И ПРОГРАММИРОВАНИЕ

Учебные материалы
для студентов специальности 6-05-0533-07
«Математика и компьютерные науки
(по профилизациям)»

В авторской редакции

Ответственный за выпуск *Е. В. Кремень*

Подписано в печать 25.08.2023. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 1,39. Уч.- изд. л. 1,31. Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике
механико-математического факультета
Белорусского государственного университета.
Пр. Независимости 4, 220030, Минск.