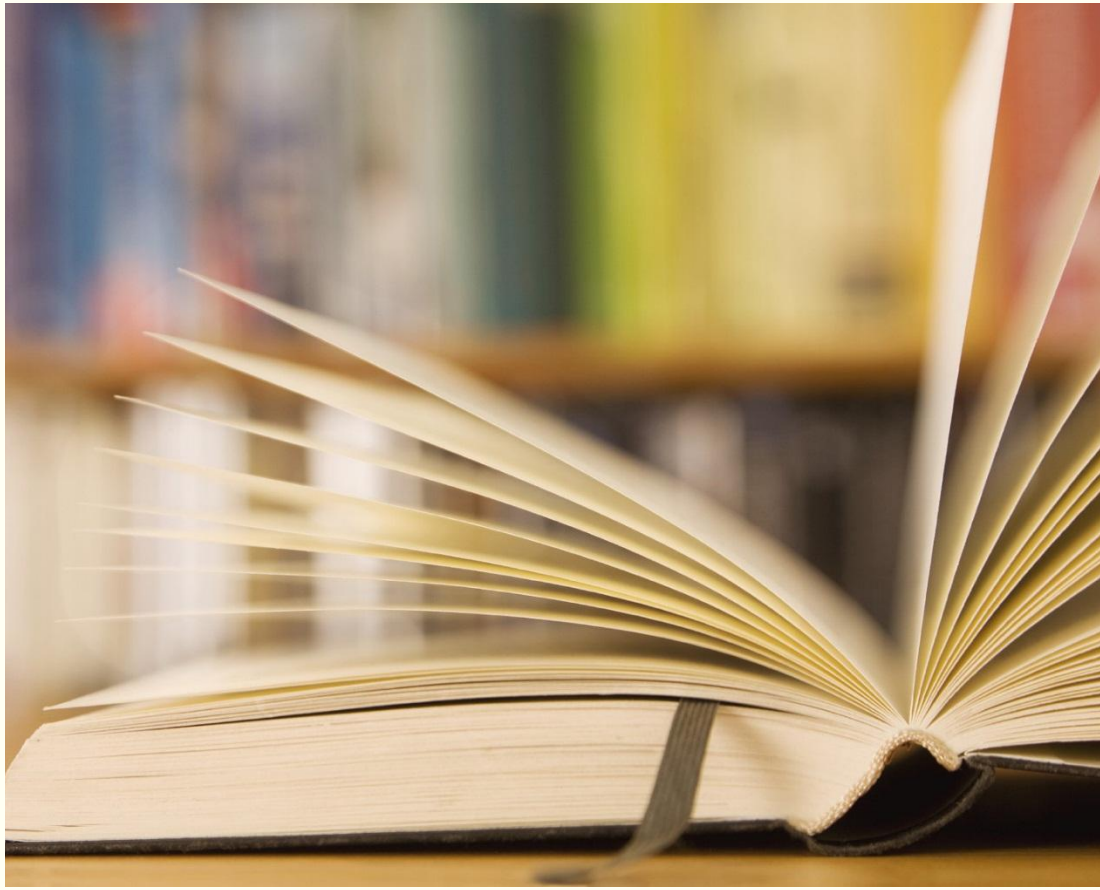
ОСНОВЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

Введение
Общие сведения
Методы проектирования
Процедуры и функции



Введение

Прошло уже более полувека со времени появления первой ЭВМ. Все это время вычислительная техника бурно развивалась. Менялась элементная база ЭВМ, росли быстродействие, объем памяти, менялись средства взаимодействия человека с машиной. Безусловно, эти изменения сказывались самым непосредственным образом на работе программиста. Определенный общепринятый способ производства чего-либо (в данном случае — программ) называют технологией. Далее мы будем говорить о *технологии программирования.*

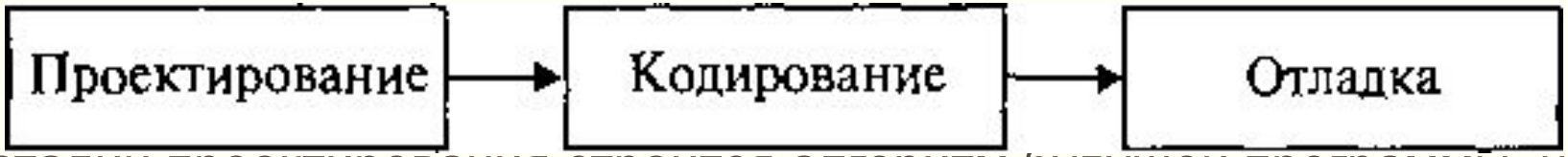
Введение

На первых ЭВМ с «тесной» памятью и небольшим быстродействием основным показателем качества программы была ее экономичность по занимаемой памяти и времени счета. Чем программа получалась короче, тем класс программиста считался выше. Такое сокращение программы часто давалось большими усилиями. Иногда программа получалась настолько «хитрой», что могла «перехитрить» самого автора. Возвращаясь через некоторое время к собственной программе, желая что-то изменить, программист мог запутаться в ней, забыв свою «гениальную идею».

Так как вероятность выхода из строя сложного технического устройства больше, чем простого, очень сложный алгоритм всегда увеличивает вероятность ошибки в программе.

Введение

В процессе изготовления программного продукта программист должен пройти определенные этапы.



На стадии проектирования строится алгоритм будущей программы, например, в виде блок-схемы. Кодирование — это составление текста программы на языке программирования. Отладка осуществляется с помощью тестов, т.е. программа выполняется с некоторым заранее продуманным набором исходных данных, для которого известен результат. Чем сложнее программа, тем большее число тестов требуется для ее проверки. Очень «хитрую» программу трудно протестировать исчерпывающим образом. Всегда есть шанс, что какой-то «подводный камень» остался незамеченным.

Введение

С ростом памяти и быстродействия ЭВМ, с совершенствованием языков программирования и трансляторов с этих языков проблема экономичности программы становится менее острой. Все более важной качественной характеристикой программ становится их простота, наглядность, надежность. С появлением машин третьего поколения эти качества стали основными.

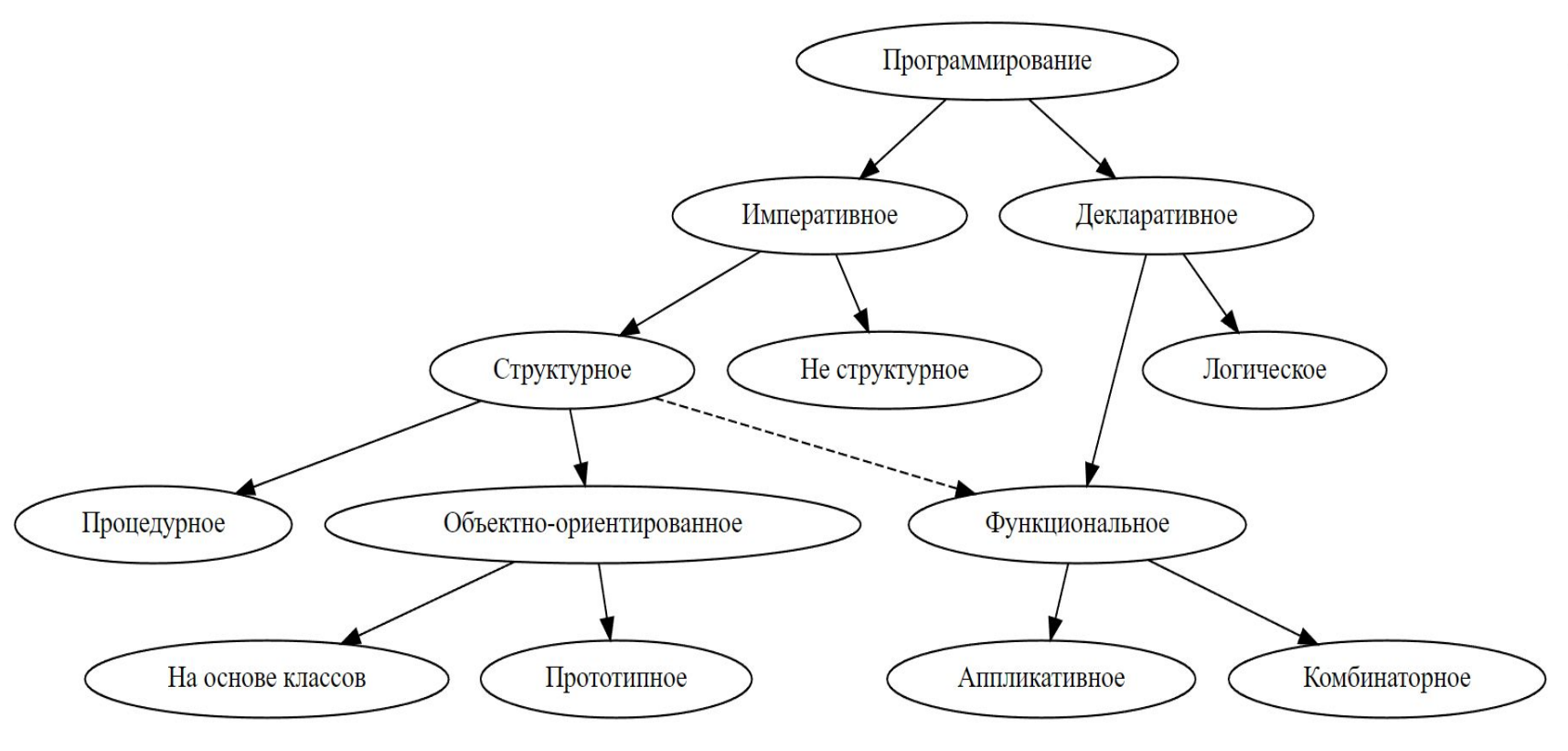
В конце 60-х — начале 70-х гг. XX столетия вырабатывается дисциплина, которая получила название структурного программирования. Ее появление и развитие связаны с именами Э.В. Дейкстры, Х.Д. Милса, Д. Е. Кнута и других ученых. Структурное программирование до настоящего времени остается основой технологии программирования. Соблюдение его принципов позволяет программисту быстро научиться писать ясные, безошибочные, надежные программы.

Парадигмы программирования

Что такое парадигма вообще?

Можно сказать, что это определенный взгляд на явления окружающего мира и представление о возможных действиях с ними.

В программировании под парадигмой принято понимать обобщение о том, как должна быть организована работа программы.



Введение

- Императивное программирование – это парадигма, основанная на составлении алгоритма действий (инструкций/команд), которые изменяют состояние (информацию/данные/память) программы.
- Декларативное программирование — это парадигма, при которой описывается желаемый результат, без составления детального алгоритма его получения.

Введение

- Неструктурное программирование - Характерно для наиболее ранних языков программирования. Из любого места программы возможен переход к любой строке.

- Функциональное программирование основано на математическом понятии функции, которая не изменяет свое окружение; это отличие функционального программирования от функций в структурных языках. Функциональная программа состоит из совокупности определений функций, которые в свою очередь представляют собой вызовы других функций и предложений, управляющих последовательностью вызовов. Каждая функция возвращает некоторое значение в вызвавшую его функцию, вычисление которой после этого продолжается; этот процесс повторяется до тех пор, пока не будет достигнут результат.

- В логическом программировании программы выражены в виде формул математической логики, и решение задачи достигается путем вывода логических следствий из них.

Введение

Объектно-ориентированное программирование

- Особое внимание уделяется данным, которые представляются в программе в виде объектов. Объекты взаимодействуют между собой с помощью механизма передачи сообщений. Задача программиста - реализовать такие объекты, при взаимодействии которых можно будет получать желаемый результат.
- ООП призвано решать более сложные и объемные задачи по сравнению с директивным программированием.
- В основе ООП лежат такие понятия как **наследование**, **полиморфизм** и **инкапсуляция**.
- Инкапсуляция предполагает, что малозначащие детали объекта скрыты. Объект, получая какую-либо команду, сам «знает» как ее обработать исходя из того, к какому классу он принадлежит.
- Все объекты являются экземплярами классов, которые по отношению друг к другу могут выступать в роли родитель-потомок. Дочерние классы наследуют свойства родительского. В случае, когда 100% наследование не требуется, выручает так называемый полиморфизм, который предполагает переопределение методов родительского класса в дочерних классах.
- Прототипное программирование — стиль ооп, при котором отсутствует понятие класса, а наследование производится путём клонирования существующего экземпляра объекта — прототипа.

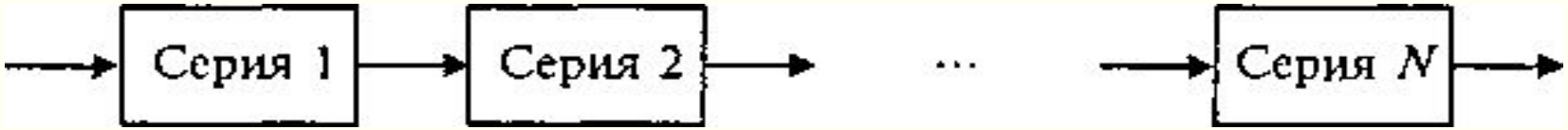
Теорема Бёма - Якопини

В основе структурного программирования лежит теорема, которая была строго доказана в теории программирования. Суть ее в том, что ***алгоритм для решения любой логической задачи можно составить только из структур «следование, ветвление, цикл».*** Их называют базовыми алгоритмическими структурами. Из предыдущих лекциях вы уже знакомы с этими структурами. По сути дела, мы и раньше во всех рассматриваемых примерах программ придерживались принципов структурного программирования.

Теорема Бёма — Якопини — положение структурного программирования, согласно которому любой исполняемый алгоритм может быть преобразован к структурированному виду, то есть такому виду, когда ход его выполнения определяется только при помощи трёх структур управления: **последовательной** (англ. *sequence*), **ветвлений** (англ. selection) и повторов или **циклов** (англ. iteration).

Базовые структуры управления

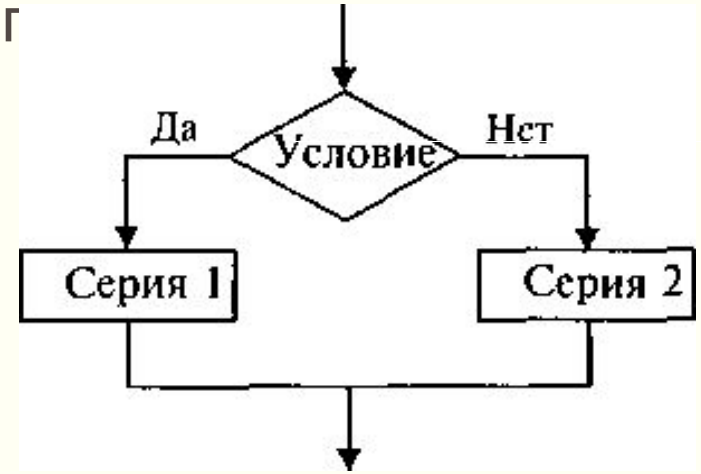
Следование — это линейная последовательность действий:



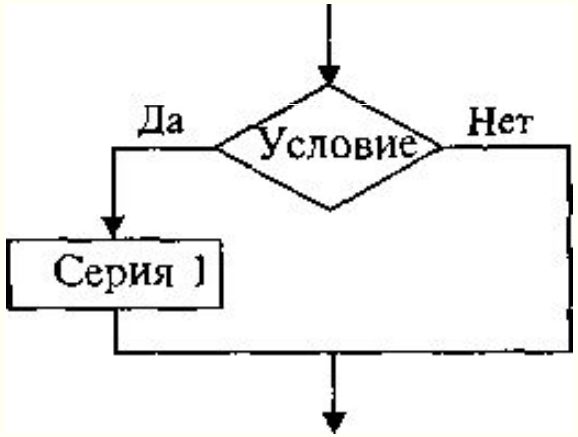
Каждый блок может содержать в себе как простую команду, так и сложную структуру, но обязательно **должен иметь один вход и один выход.**

Базовые структуры управления

Ветвление — алгоритмическая альтернатива. Управление передается одному из двух блоков в зависимости от истинности или ложности условия. Затем происходит выход на общее



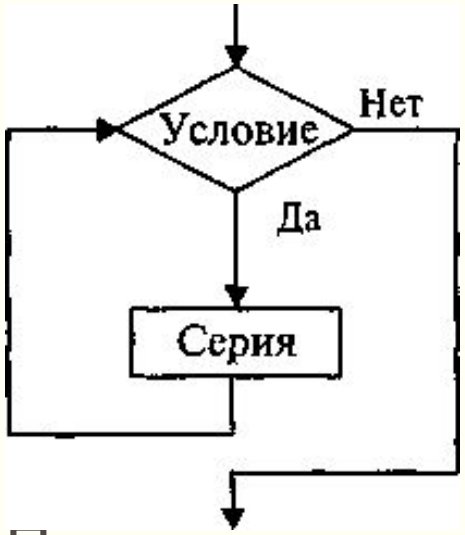
если условие
то серия 1
иначе (серия 2)
кц



Базовые структуры управления

Цикл — повторение некоторой группы действий по условию.

Цикл с предусловием (цикл-пока):



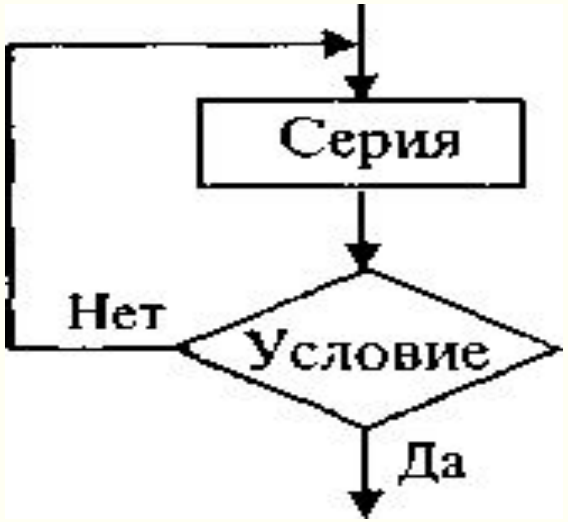
пока условие
повторять
нц
серия
кц

Пока условие истинно, выполняется серия, образующая тело цикла.

Базовые структуры управления

Цикл — повторение некоторой группы действий по условию.

Цикл с постусловием (цикл-до):



повторять

серия

до условие

Здесь тело цикла предшествует условию цикла. Тело цикла повторяет свое выполнение, если условие ложно. Повторение кончается, когда условие станет истинным.

Базовые структуры управления

Теоретически необходимым и достаточным является лишь первый тип цикла — цикл с предусловием. Любой циклический алгоритм можно построить с его помощью. Это более общий вариант цикла, чем цикл-до. В самом деле, тело цикла-до хотя бы один раз обязательно выполнится, так как проверка условия происходит после завершения его выполнения. А для цикла-пока возможен такой вариант, когда тело цикла не выполнится ни разу. Поэтому в любом языке программирования можно было бы ограничиться только циклом-пока. Однако в ряде случаев применение цикла-до оказывается более удобным, и поэтому он используется.

Оператор GOTO

- **Оператор goto** — это **оператор управления потоком выполнения программ**, который заставляет центральный процессор выполнить переход из одного участка кода в другой (осуществить прыжок). Другой участок кода идентифицируется с помощью **лейбла**. Например:

```cpp
#include <iostream>
#include <cmath> // для функции sqrt()

int main()
{
    double z;
tryAgain: // это лейбл
    std::cout << "Enter a non-negative number: ";
    std::cin >> z;

    if (z < 0.0)
        goto tryAgain; // а это оператор goto

    std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;
    return 0;
}
```

Оператор GOTO

В этой программе пользователю предлагается ввести неотрицательное число. Однако, если пользователь введет отрицательное число, программа, используя оператор goto, выполнит переход обратно к лейблу tryAgain. Затем пользователю снова нужно будет ввести число. Таким образом, мы можем постоянно запрашивать у пользователя ввод числа, пока он не введет корректное число.

В целом, программисты избегают использования оператора goto в большинстве высокоуровневых языков программирования. Основная проблема с ним заключается в том, что он позволяет программисту управлять выполнением кода так, что точка выполнения может прыгать по коду произвольно. А это, в свою очередь, создает то, что опытные программисты называют «спагетти-кодом».

Спагетти-код — это код, порядок выполнения которого напоминает тарелку со спагетти (всё запутано и закручено), что крайне затрудняет следование порядку и понимание логики выполнения такого кода.

Как говорил один известный специалист в информатике и программировании, Эдсгер Дейкстра: «Качество программистов — это уменьшающаяся функция плотности использования операторов goto в программах, которые они пишут».

Правило: Избегайте использования операторов goto, если на это нет веских причин.

Взаимодействие базовых структур

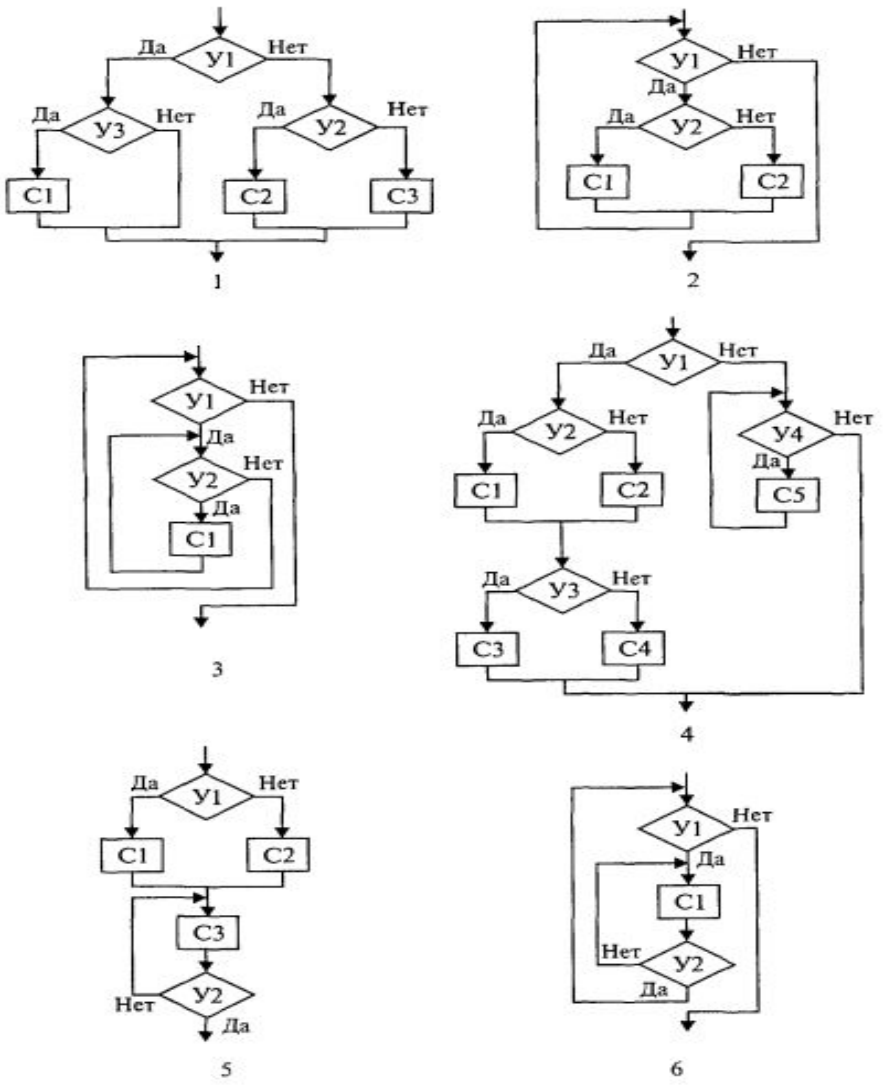
Сложный алгоритм состоит из соединенных между собой базовых структур. Соединяться эти структуры могут двумя способами: *последовательным и вложенным*. Эта ситуация аналогична тому, что мы наблюдаем в электротехнике, где любая сколь угодно сложная электрическая цепь может быть разложена на последовательно и параллельно соединенные участки.

Вложенные алгоритмические структуры не являются аналогом параллельно соединенных проводников. Здесь больше подходит аналогия с матрешками, помещенными друг в друга. Если блок, составляющий тело цикла, сам является циклической структурой, то, значит, имеют место вложенные циклы. В свою очередь, внутренний цикл может иметь внутри себя еще один цикл и т.д. В связи с этим вводится представление о глубине вложенности циклов. Точно так же и ветвления могут быть вложенными друг в друга.

Взаимодействие базовых структур

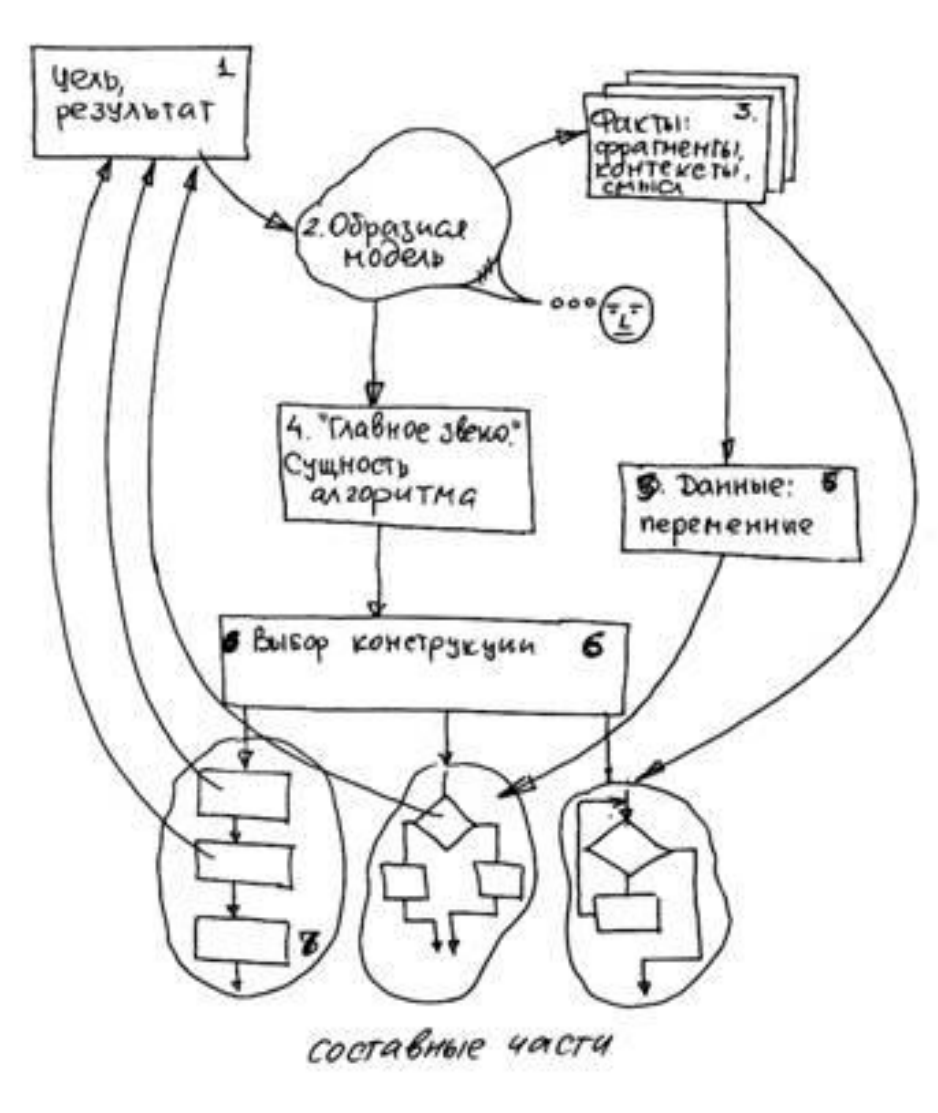
Структурный подход требует соблюдения стандарта в изображении блок-схем алгоритмов. Чертить их нужно так, как это делалось во всех приведенных примерах. Каждая базовая структура должна иметь один вход и один выход. Нестандартно изображенная блок-схема плохо читается, теряется наглядность алгоритма. Вот несколько примеров структурных блок-схем алгоритмов. Такие блок-схемы легко читаются. Их структура хорошо воспринимается зрительно. Структуре каждого алгоритма можно дать название. У приведенных на рис. блок-схем следующие названия:

1. Вложенные ветвления. Глубина вложенности равна единице.
2. Цикл с вложенным ветвлением.
3. Вложенные циклы-пока. Глубина вложенности — единица.
4. Ветвление с вложенной последовательностью ветвлений на положительной ветви и с вложенным циклом-пока на отрицательной ветви
5. Следование ветвления и цикла-до.
6. Вложенные циклы. Внешний — цикл-пока, внутренний — цикл-до.



Разработка сверху вниз и снизу вверх

Технология структурного программирования в самой краткой формулировке есть нисходящее проектирование, т.е. выстраивание текста программы, точнее алгоритмической компоненты, от общего к частному, от внешней конструкции к внутренней. Естественно, что надо знать, из чего выстраивать. В идеале, у опытного программиста действительно очередная нужная конструкция появляется «из головы». Но это не значит, что он не имеет общего плана действий и обобщенного представления процесса, который реализуется проектируемой программой.

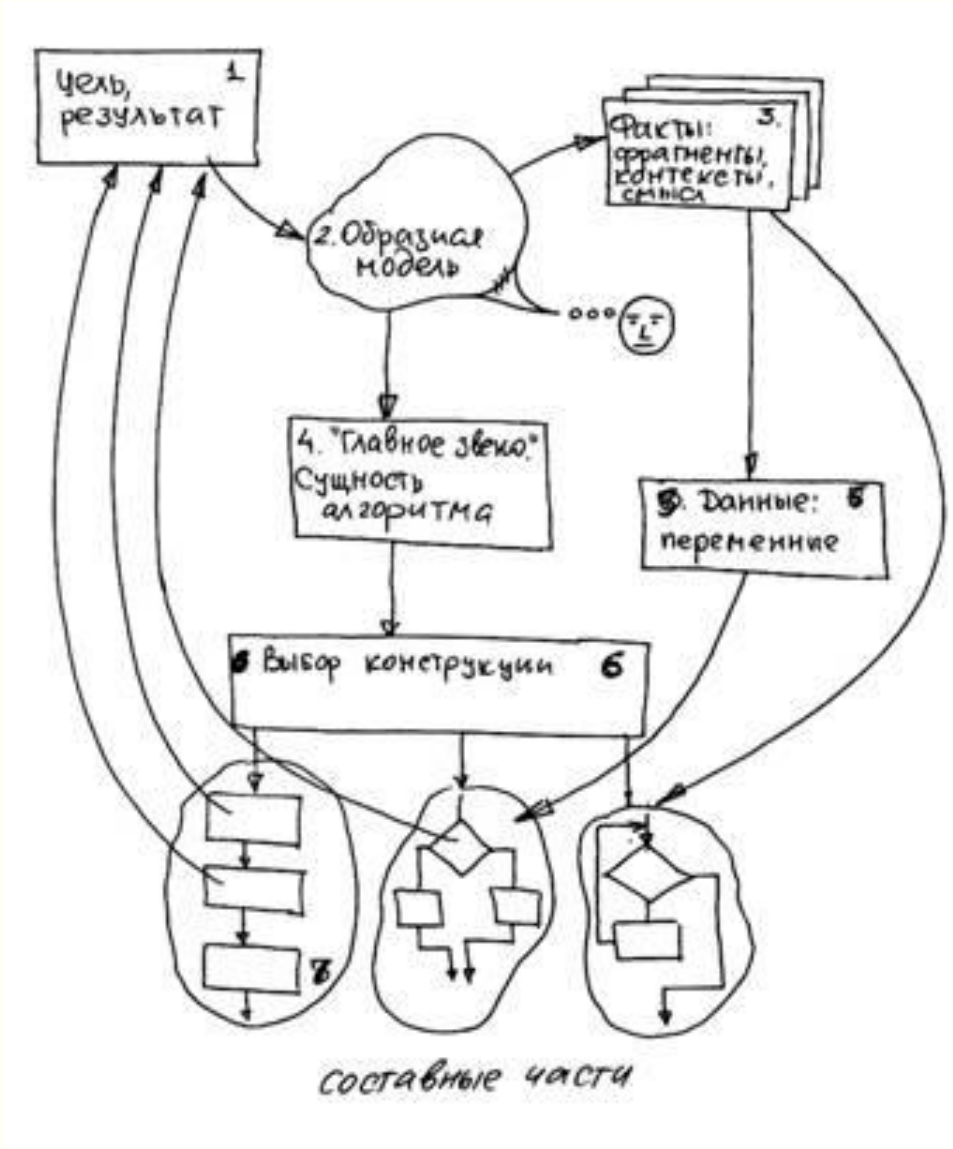


Разработка сверху вниз и снизу вверх

Именно поэтому в технология программирования была обозначена как заключительный этап выстраивания программы из имеющегося набора фрагментов. Перед этим необходимо пройти другие этапы:

- формулировка целей (результатов) работы программы;
- образное представление процессы ее работы (образная модель);
- выделение из образной модели фрагментов: определение переменных и их смыслового наполнения, стандартных программных контекстов.

Встроим в общую схему процесса проектирования самое трудное направление «движения» при построении программы – от общего к частному. И тогда получим примерно такую картину.



Разработка сверху вниз и снизу вверх

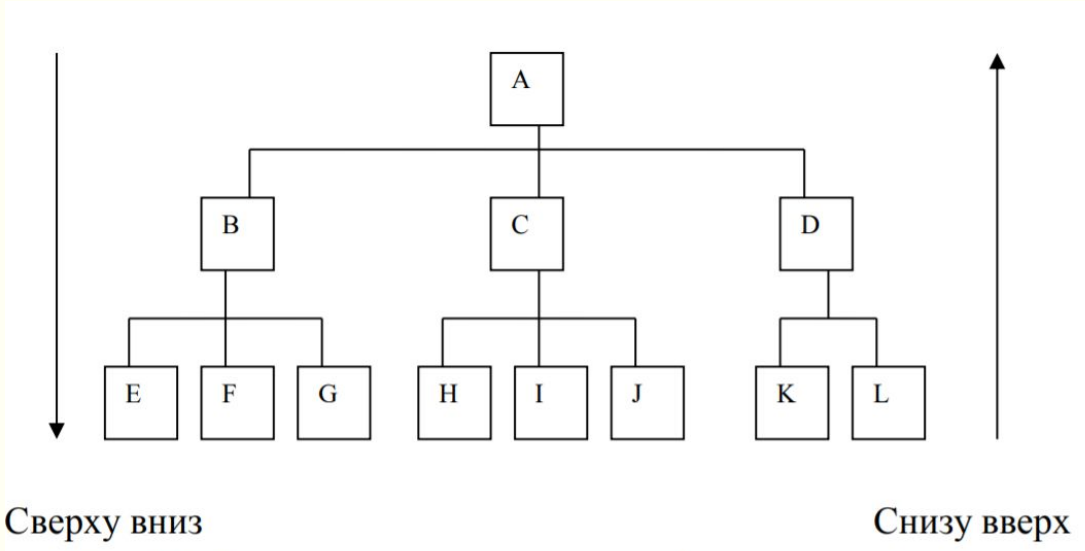
- 1. Исходным состоянием процесса проектирования является более или менее точная формулировка цели алгоритма, или результата, который должен быть получен при его выполнении. Формулировка, само собой, производится на естественном языке.
- 2. Создается образная модель происходящего процесса, используются графические и какие угодно способы представления, образные «картинки», позволяющие лучше понять выполнение алгоритма в динамике;
- 3. Выполняется сбор фактов, касающихся любых характеристик алгоритма, и попытка их представления средствами языка. Такими фактами является наличие определенных переменных и их «смысл», а также соответствующих им программных контекстов. Понятно, что не все факты удастся сразу выразить в виде фрагментов программы, но они должны быть сформулированы хотя бы на естественном языке;

Разработка сверху вниз и снизу вверх

- 4. В образной модели выделяется наиболее существенная часть – «главное звено», для которой подбирается наиболее точная словесная формулировка;
- 5. Производится определение переменных, необходимых для формального представления данного шага алгоритма и формулируется их «смысл»;
- 6. Выбирается одна из конструкций - *простая последовательность действий, условная конструкция* или *цикл*. Составные части выбранной формальной конструкции (например, условие, заголовок цикла) должны быть переписаны в словесной формулировке в виде цели или результата, которые должны давать эти части алгоритма.
- 7. Для оставшихся неформализованных частей алгоритма (в словесной формулировке) - перечисленная последовательность действий повторяется. Обычно разработка образного представления программы опережает ее «выстраивание», поэтому следующим этапом для неформализованной части алгоритма может быть п.4 (в лучшем случае, при его проработке в образной модели) или п.1-3. В любом случае для вложенных конструкций мы возвращаемся на предыдущие этапы проектирования.

Разработка сверху вниз и снизу вверх

Любая система состоит из частей, каждую из которых также можно разложить на составные части и т. д. Например, университет делится на факультеты, факультеты включают кафедры и курсы, разделенные на
лаборатории и группы, которые, в конечном счете, состоят из сотрудников и студентов.
Структуру самых разнообразных систем: технических, организационных, биологических, грамматических и т. п. часто изображают в виде дерева, обычно с корнем наверху Таким же образом можно представить любой алгоритм или программу, разбивая их на более мелкие алгоритмы, блоки и т. д. до команд. Каждый узел дерева обозначает некоторый алгоритм. Отходящие от него вниз линии ведут к составным частям этого алгоритма. Алгоритм A включает в себя алгоритмы B, D, C; алгоритм D, в свою очередь,



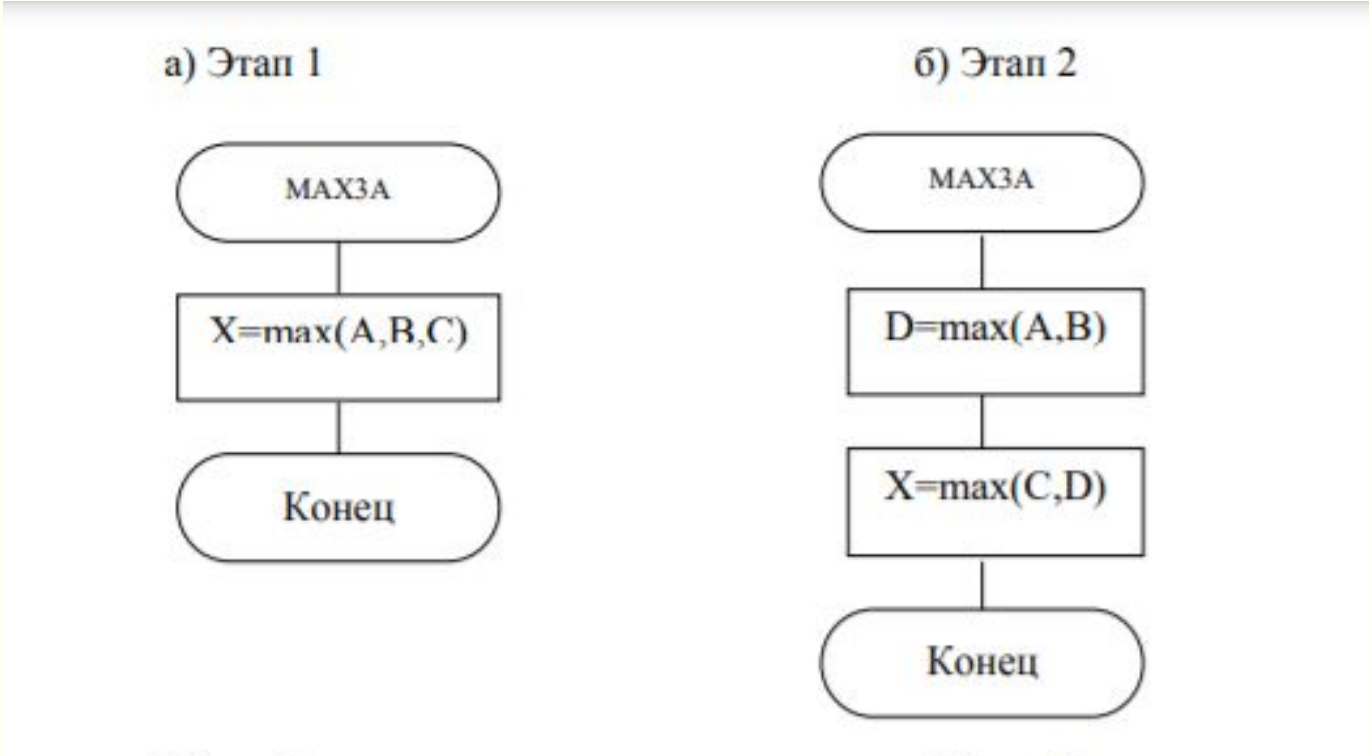
Разработка сверху вниз и снизу вверх

Разработка сверху вниз начинается от главной цели: на каждом этапе разработки решаемая задача (поставленная цель) разбивается на более простые подзадачи (подцели), с которыми затем поступают таким же образом.

Так, при написании книги или сочинения сначала составляется план: книга разбивается на части; а затем уже уточняются детали, т. е. пишутся планы и тексты этих частей. На первом этапе проектируемый алгоритм представляется в виде одного блока. Затем определяется структура этого блока (например, выбирается одна из базовых структур структурного программирования).

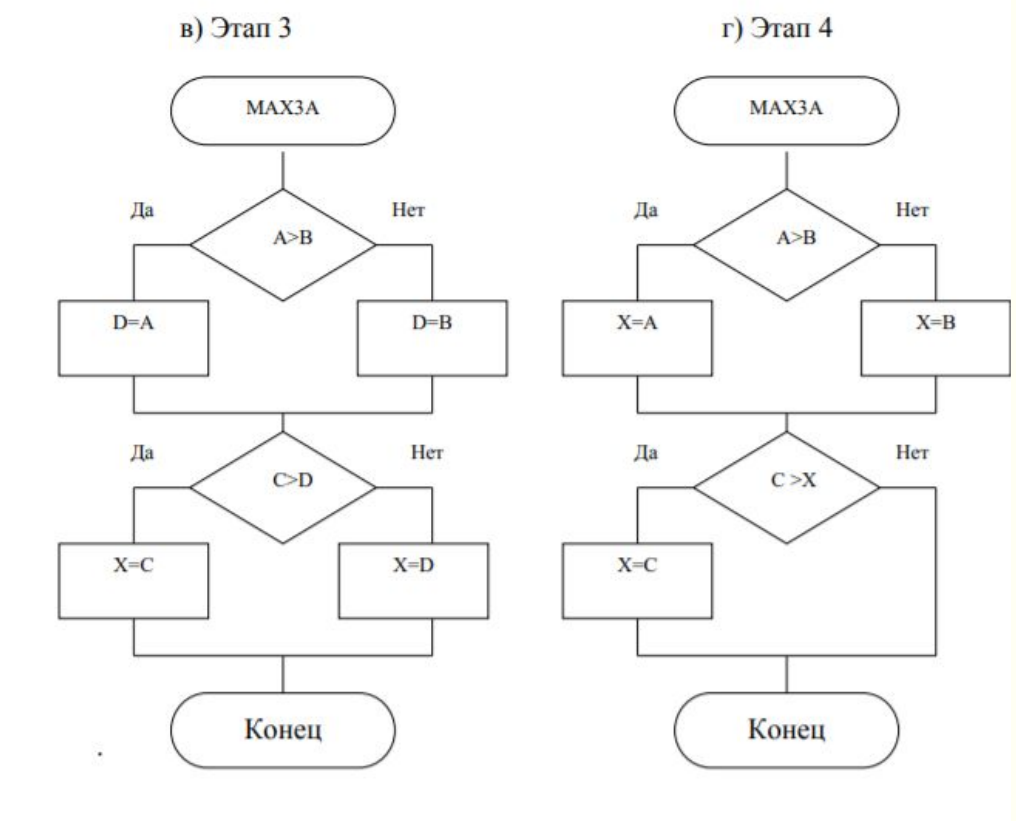
Таким образом, исходный алгоритм разбивается на части. Далее разработка продолжается аналогично: каждый блок разбивается на более мелкие действия, пока весь алгоритм не будет разложен на достаточно простые операции, "понятные" процессору (имеющиеся в выбранном языке программирования).

Пример. Присвоить X максимум из 3х чисел (A,B,C)



Этап 1 – Исходный алгоритм
Этап 2 – Определение структуры алгоритма

Пример. Присвоить X максимум из 3х чисел (A,B,C)



Этап 3 – Определение структуры блоков
Этап 4 – Устранение вспомогательной переменной

Пример реализации процедурного подхода на ЯП C#

```
1   using System;
2
3   namespace MasMethods
4   {
5       class Program
6       {
7
8           static void ListInitRand(int[] Array)... // Метод инициализации массива случайными числами (процедура)
18          static int[] SortList(int[] Array)... // Метод сортировки массива (функция)
33          static void SwapTwoNums (ref int num1, ref int num2)...  // Метод перестановки значений двух переменных (процедура)
40          static int BinarySearch(int[] Array, int item)...  // Метод бинарного поиска (функция)
59
60          static void WriteList(int[] Array)... // Метод вывода массива на экран (процедура)
65          static void Main(string[] args) // Метод выполнения программы, где вызываются методы
66          {
67              int size = Int32.Parse(Console.ReadLine());
68              int[] ourArray = new int[size];
69              ListInitRand(ourArray);
70              WriteList(ourArray);
71              Console.WriteLine("***********");
72              int[] newArray = SortList(ourArray);
73              WriteList(newArray);
74              Console.WriteLine("***********");
75              Console.WriteLine(BinarySearch(newArray, 6));
76
```

Пример реализации процедурного подхода. Вложенность методов

```
19      static int[] SortList(int[] Array) // Метод сортировки массива (функция)
20      {
21          for (int i = 0; i < Array.Length; i++)
22          {
23              for (int j = i + 1; j < Array.Length; j++)
24              {
25                  if (Array[i] > Array[j])
26                  {
27                      SwapTwoNums(ref Array[i], ref Array[j]); // В методе сортировки вызывается метод перестановки значений переменных
28                  }
29
30              }
31          }
32          return Array;
33      }
34      static void SwapTwoNums (ref int num1, ref int num2)  // Метод перестановки значений двух переменных (процедура)
35      {
36          int temp = num1;
37          num1 = num2;
38          num2 = temp;
39
40      }
```