Основы конструирования программ

Преподаватель:

к.т.н., доцент кафедры Инженерной психологии и эргономики
Меженная Марина Михайловна
mezhennaya@bsuir.by
а 606-2

Лекция 2: Методология структурного программирования

План лекции:

1.Спагетти-код: использование оператора безусловного перехода goto.
2.Методология структурного программирования: основные принципы.
3.Методология структурного программирования: создание модульной структуры программы.
4.Методология структурного программирования: нисходящее проектирование алгоритма.
5.Методология структурного программирования: восходящее проектирование алгоритма.

Неструктурные языки программирования

Изначально программы создавались так, чтобы задействовать минимум основной памяти и решить требуемую задачу за кратчайшее время.

Особенностью таких программ являлось множественное использование оператора безусловного перехода **goto** (аналог – **jump** в языках ассемблера).

Подобные программы – слабо структурированные и трудные для понимания – получили название **спагетти-код**: ход выполнения такой программы похож на миску спагетти, то есть извилистый и запутанный.

Спагетти-код – самый распространённый антипаттерн программирования.

Пояснение: Паттерны проектирования

Паттернами (шаблонами) проектирования (Design Patterns) называют решения часто встречающихся проблем в области разработки программного обеспечения.

Паттерны проектирования не являются готовыми решениями, которые можно трансформировать непосредственно в код, а представляют общее описание решения проблемы, которое можно использовать в различных ситуациях.

Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем.

Пояснение: Оператор безусловного перехода goto

Оператор goto осуществляет переход к определённой точке программы, обозначенной номером строки либо меткой. Далее исполняются операторы программы, идущие в тексте непосредственно после метки.

```
goto label;

// Блок кода

label: оператор;

// Блок кода
```

Пояснение: Оператор безусловного перехода goto

Некоторые способы применения goto могут создавать проблемы с логикой исполнения программы:

Переход в точку программы, расположенную после инициализации какой-либо переменной, приведёт к тому, что для этой переменной будет использовано некоторое случайное значение, которое находилось в памяти.

```
for(int i=0; i<n; i++) {
   if (condition) goto label;
    // Блок кода
}

int number = 25;

label: оператор;
int temp = number;
// Блок кода
```

Пояснение: Оператор безусловного перехода goto

Некоторые способы применения goto могут создавать проблемы с логикой исполнения программы:

Передача управления внутрь тела цикла (процедуры или функции) приводит к пропуску кода инициализации цикла (выделение памяти под локальные переменные) и первоначальной проверки условия.

```
for(int i=0; i<n; i++) {
   if (condition) goto label;
    // Блок кода
}

for(int j=0; j<m; j++) {
   // Блок кода
   label: оператор;
}
```

Пояснение: Оператор безусловного перехода goto

Оператор goto в языках высокого уровня является объектом критики, поскольку чрезмерное его применение приводит к созданию нечитаемого «спагетти-кода».

Формально доказано (теорема Бёма — Якопини), что **применение goto не является обязательным**, то есть не существует такой программы с goto, которую нельзя было бы переписать без него с полным сохранением функциональности (однако, возможно, с потерей эффективности).

Пояснение: Оператор безусловного перехода goto

В практическом программировании применение goto считается допустимым в одном случае – для выхода из нескольких вложенных циклов сразу.

```
int matrix[n][m];
int value;
// Блок кода

for(int i=0; i<n; i++) {
    for (int j=0; j<m; j++) {
        if (matrix[i][j] == value) {
          goto stop;
        }
        // Блок кода
    }
}

stop: ;
```

Пояснение: Оператор безусловного перехода goto

Альтернатива:
применение
вспомогательных
переменных-флагов и
условных операторов.

```
int matrix[n][m];
int value;
bool flag = false;
// Блок кода

for(int i=0; i<n; i++) {
  for (int j=0; j<m; j++) {
       if (matrix[i][j] == value) {
          flag = true;
          break;
       }
       // Блок кода
   }
   if (flag) break;
}
```

Методология структурного программирования

Спагетти-код может быть отлажен и работать правильно и с высокой производительностью, но он крайне сложен в сопровождении и развитии. Доработка спагетти-кода для добавления новой функциональности иногда несет значительный потенциал внесения новых ошибок.

Исходя из проблем, связанных с программами спагетти-кода, программисты 70-х годов Э.Дейкстра и Н.Вирт разработали строгие правила разработки программ, которые получили название **“Методология структурного программирования”**.

Принципы структурного программирования

1.Программа (алгоритм) должна разделяться на независимые части, называемые **модулями**. Модуль – это последовательность логически связанных операций, оформленных как отдельная часть программы.

2.Модуль имеет **одну входную** и **одну выходную** точку (в отличие от программ с множественным использованием оператора goto).

3.В модуле используются только три базовые управляющие конструкции (**следование, ветвление, цикл**).
Теорема Бёма — Якопини (1965 год), или «Теорема о структурном программировании»: Любая программа, заданная в виде блок-схемы, может быть представлена с помощью трех управляющих структур: последовательность, ветвление, цикл.

Принципы структурного программирования

4.В программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом. Никаких других средств управления последовательностью выполнения операций не предусматривается (т.е. **от использования оператора безусловного перехода goto следует отказаться**).

5.Повторяющиеся фрагменты программы можно оформить в виде подпрограмм (процедур и функций). Таким же образом (в виде подпрограмм) можно оформить логически целостные фрагменты программы, **даже если они не повторяются**.

6.Разработка программы ведётся пошагово, методом «сверху вниз» (**нисходящее проектирование**).

Методология структурного программирования

Использование модулей имеет следующие преимущества:

- возможность создания программы несколькими программистами;
- простота проектирования и последующей модификаций программы;
- возможность использования готовых библиотек наиболее употребительных модулей.

Достоинства структурного программирования

- повышается надежность программ (благодаря хорошему структурированию при проектировании программа легко поддается тестированию и не создает проблем при отладке);

- повышается эффективность программ (структурирование программы позволяет легко находить и корректировать ошибки, а отдельные подпрограммы можно переделывать (модифицировать) независимо от других);

- уменьшается время и стоимость программной разработки;

- улучшается читабельность программ.

Конструирования ПО в контексте методологии структурного программирования

Конструирование ПО =
детальное проектирование + программирование

Для алгоритмических языков программирования:

Детальное проектирование =
анализ задачи + разработка алгоритма

Конструирование ПО в соответствии с методологией структурного программирования:

анализ задачи → создание модульной структуры ПО,
разработка алгоритма → метод нисходящего проектирования.

Конструирования ПО в контексте методологии структурного программирования: анализ задачи

Анализ задачи – это исследование объектов или явлений, путем изучения составляющих его элементов.

Анализ задачи при разработке ПО включает:

- установление входных и выходных данных;
- разработку основных решений, приводящих от входных к выходным данным;
- выделение модулей, необходимых для выполнения задачи (реализация методологии структурного программирования);
- определение методов частных решений для выделенных модулей.

Основным результатом анализа задачи является **создание модульной структуры ПО.**

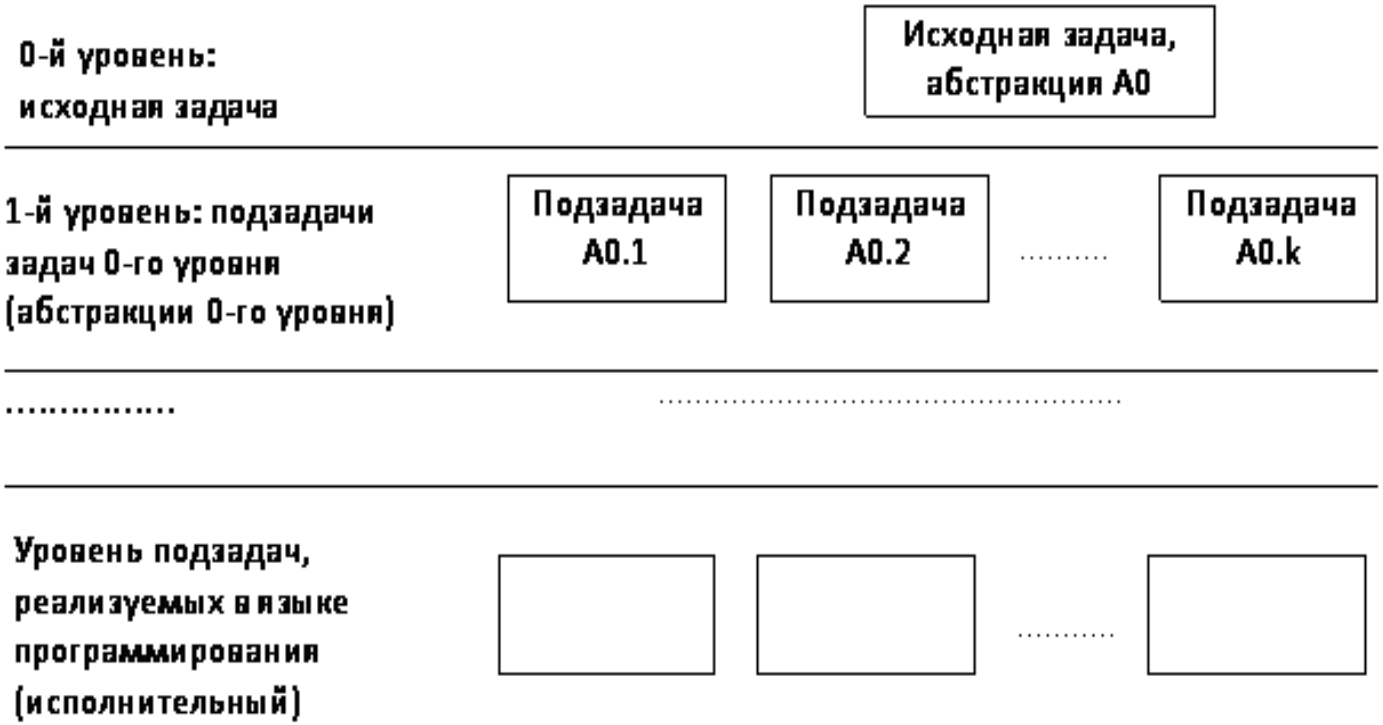
Конструирования ПО в контексте методологии структурного программирования: разработка алгоритма

Для построения структурированных блок-схем в рамках методологии структурного программирования была разработана специальная технология – **нисходящее проектирование**, которая состоит в **пошаговой детализации (декомпозиции, разложении)** задачи на подзадачи.

Пошаговая детализация представляет собой процесс дробления задачи на подзадачи, установления логических связей между ними. После этого переходят к уточнению выделенных подзадач.

Этот процесс детализации продолжается до уровня, позволяющего достаточно легко реализовать подзадачу на выбранном языке программирования.

Разработка алгоритма: нисходящее проектирование



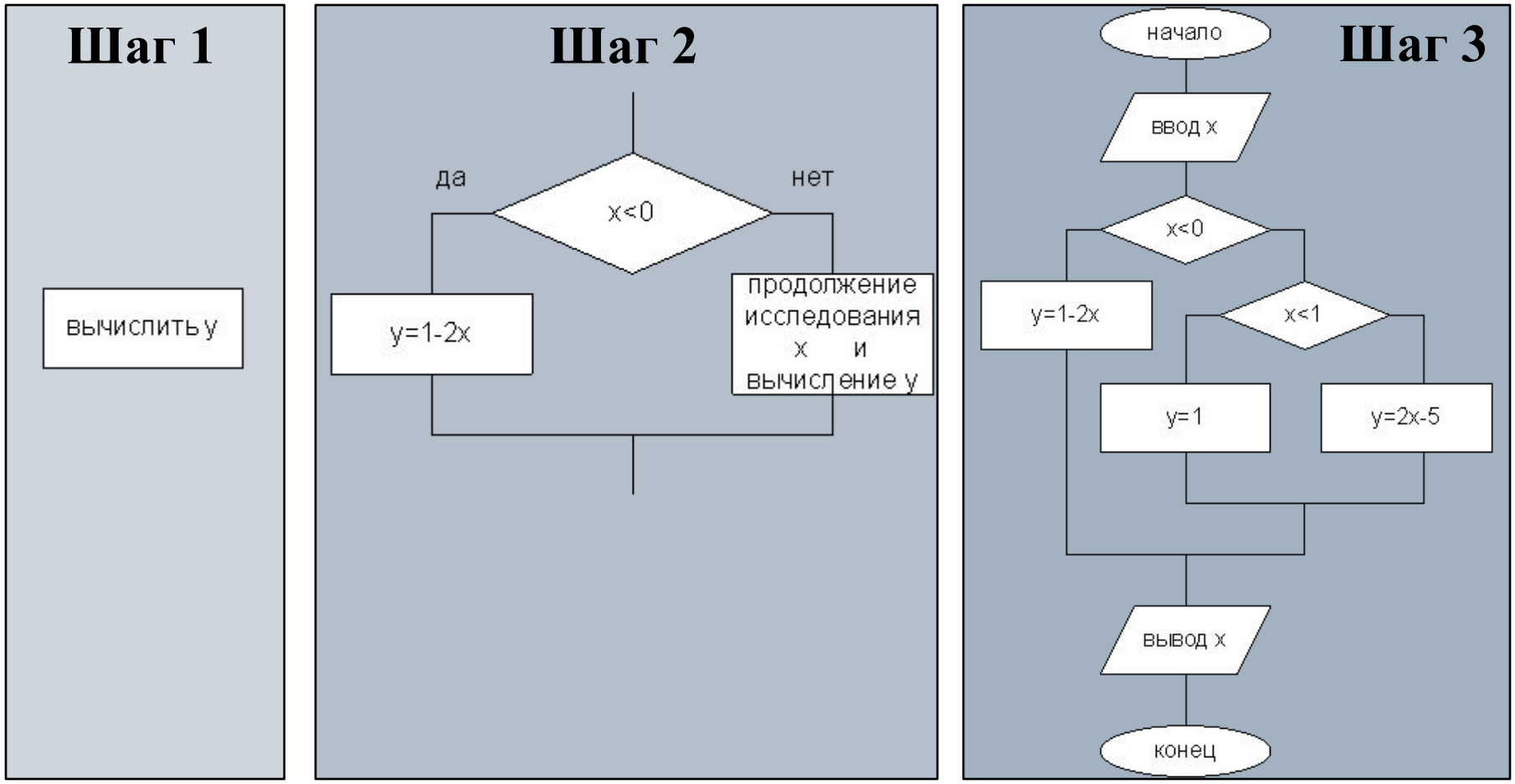
Общая схема нисходящей разработки

Разработка алгоритма: нисходящее проектирование

При пошаговой детализации используется **принцип замещения**, который состоит в замене любого функционального блока (прямоугольника) блок-схемы некоторой базовой структурой (следование, ветвление, повторение).

Пример. Построить структурированную блок-схему алгоритма для вычисления функции:

$$y=\begin{cases} 1-2x, & \text{если } x<0; \\ 1, & \text{если } 0<=x<1; \\ 2x-5, & \text{если } x>=1. \end{cases}$$

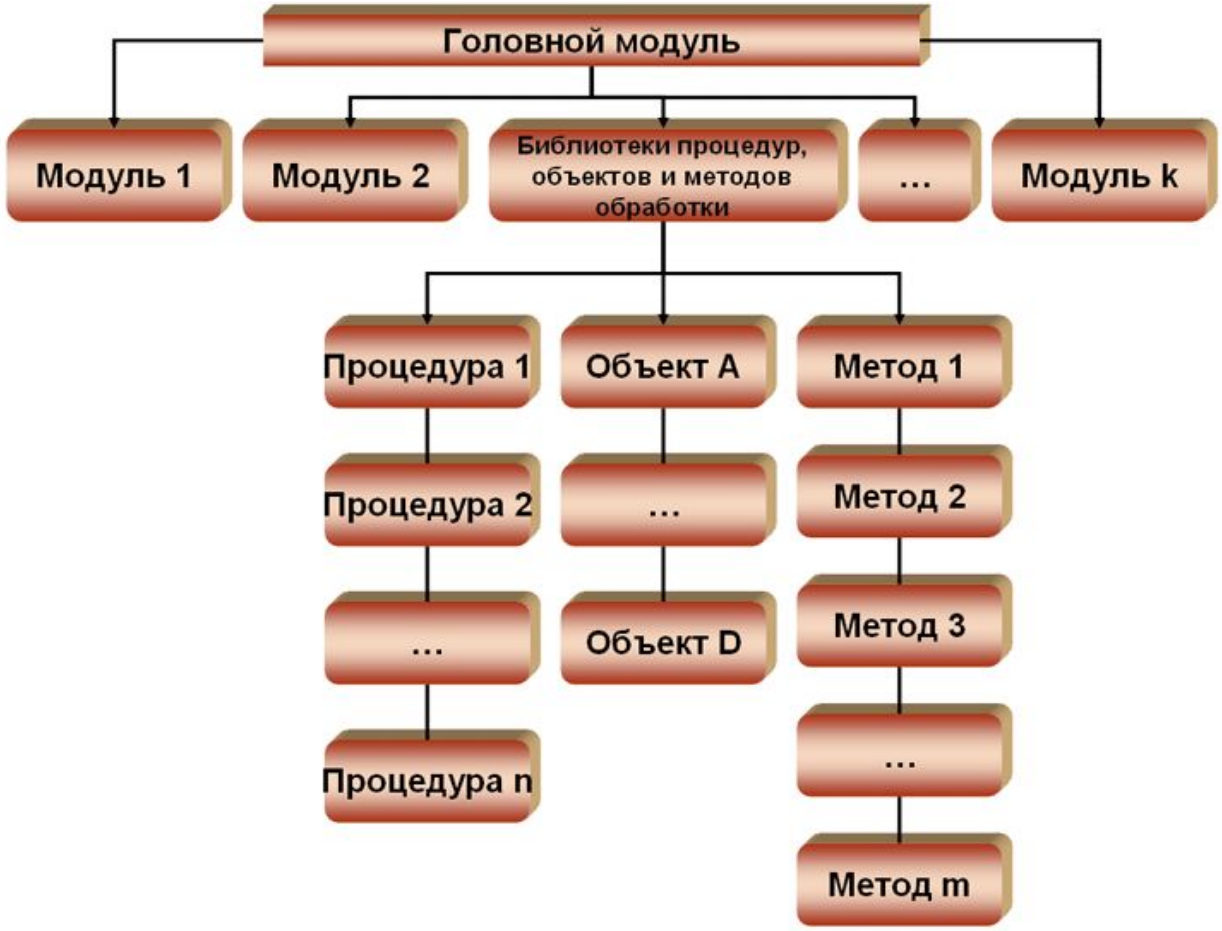


Разработка алгоритма: нисходящее проектирование

Алгоритмы большой сложности обычно представляются с помощью схем двух видов:

- обобщенной схемы алгоритма - раскрывающей общий принцип функционирования алгоритма и основные логические связи между отдельными модулями на уровне типовых процессов обработки информации (ввод и редактирование данных, вычисления, печать результатов и т.п.);
- детальной схемы алгоритма - представляющей содержание каждого элемента обобщенной схемы с использованием управляющих структур в блок-схемах алгоритма с использованием графических элементов согласно требованиям структурного программирования.

Типовая структура программы, разработанной в соответствии со структурной методологией



Разработка и отладка программы, созданной в соответствии с принципами структурного программирования

Можно разработать тест основной программы таким образом, чтобы вместо каждого связного логического фрагмента вставлялся вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих подпрограмм, в программу вставляются фиктивные части — **заглушки**, которые, говоря упрощенно, ничего не делают.

Затем заглушки заменяются или дорабатываются до настоящих полнофункциональных фрагментов (модулей).

Разработка заканчивается тогда, когда не останется ни одной заглушки. Полученная программа проверяется и отлаживается.

Разработка и отладка программы, созданной в соответствии с принципами структурного программирования

Такой процесс отладки дает на каждом уровне два важных результата:

- возможность отладки уже готовых частей программы;
- возможность отладки взаимодействия подзадач, т.е. логики передачи данных между ними.

При этом то, что уже отлажено, далее меняться не должно, и все усилия можно сосредоточить на раскрытии еще не решенных подзадач.

Разработка алгоритма: восходящее проектирование

Восходящее проектирование – методика разработки программ, при которой крупные блоки собираются из ранее созданных мелких блоков.

Случаи восходящего проектирования:

1.Разработчик ясно представляет направление поиска, но не знает заранее, как далеко он сможет продвинуться к цели.

2.Нет возможности предвидеть объем ресурсов для достижения того или иного результата.

3.Разработка не поддается детальному планированию, она ведется методом проб и ошибок.