Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Информатика и программирование

Расчётно-графическое задание

Методические указания

Новосибирск

20017

Методические указания содержат варианты заданий и указания к выполнению расчётно-графического задания по курсу Информатика и программирование на языке C# в системе программирования Microsoft Visual Studio, для студентов, обучающихся по направлению 380305 бизнес - информатика дневная и заочная формы обучения.

Составитель: к.ф.-м.н., доц. М.Г.Зайцев

Рецензент: к.т.н., доц. В.Г.Кобылянский

Работа подготовлена кафедрой теоретической и прикладной информатики

Новосибирский технический университет, 20017 г.

Оглавление

Оглавление 3
Тема 4
Цель 4
Задание 4
Спецификация программы 4
Варианты заданий 9
Порядок выполнения 14
Содержание отчета 15
Основные сведения 15
Технология структурного программирования 15
Структурные методы анализа и проектирования ПО 17
Модульное программирование 20
Базовые управляющие структуры структурного программирования 25
Проектирование и тестирование программы 27
Методы 33
Пример выполнения. Вариант задания 29*. Варианты с 1 по 14 выполняются по аналогии. 39
Схема иерархии логических модулей. 40
HIPO-диаграммы для логических модулей 40
Пример структуры консольного приложения для перевода десятичного действительного числа в р-ичное представление представленное строкой. 43
Пример выполнения. Вариант задания 30*. Варианты с 15 по 28 выполняются по аналогии 45
Схема иерархии логических модулей. 45
HIPO-диаграммы для логических модулей 46
Пример структуры консольного приложения для перевода р-ичного числа в его десятичный эквивалент. 51
Основная литература 53

Тема

Технология структурного программирования

Цель

Сформировать практические навыки разработки консольных приложений под Windows:

- проектирования программ методами структурного программирования;
- реализации программного проекта средствами языка программирования C# в среде программирования Microsoft Visual Studio;
- тестирования программ.

Задание

Спроектируйте и реализуйте консольное приложение, для преобразования числа из одного формата «входного» в другой формат «выходной» в соответствии с приведённой ниже спецификацией. Входной и выходной форматы определены в варианте задания.

В качестве «входного» и «выходного» форматов взяты десятичное и представления действительного числа в системе счисления с основанием p.

Спецификация программы

Для описания требований к приложению варианты заданий разбиты на два подмножества.

- Для первого подмножества вариантов заданий в качестве «входного формата» выбрано десятичное представление числа, в качестве «выходного» - представление числа в системе счисления с основанием p.

- Для второго подмножества вариантов заданий в качестве «входного формата» выбрано представление числа в системе счисления с основанием p, в качестве «выходного» - десятичное представление числа.

Для обоих подмножеств значение p зависит от номера варианта задания.

Спецификация приложения для вариантов с 1 по 14 и 29*

1. Приложение работает в режиме диалога с пользователем. Пользователь вводит основание системы счисления p и десятичное действительное число с клавиатуры. Приложение формирует строку, содержащую представление введённого десятичного числа в системе счисления с основанием p, и выводит её на экране монитора.

Примерный результат работы приложения для варианта 29* представлен ниже на Рисунок 1.

```
//-----------------------------------------------------------------
   Приложение для перевода десятичных действительных чисел
   в систему счисления с основанием p от 2 до 16
//-----------------------------------------------------------------
  Введите основание с. сч. p - число в диапазоне 2..16: 16
//-----------------------------------------------------------------
  Введите действительное число: -175.875
//-----------------------------------------------------------------
//-----------------------------------------------------------------
  16  -ичное представление: -AF.E  десятичного числа:  -175.8750
//-----------------------------------------------------------------
```

Рисунок 1. Ошибки ввода отсутствуют. Десятичное число переводится в 16-ичную систему счисления.

2. В приложении необходимо обеспечить контроль ввода исходных данных. Так при вводе недопустимых символов в изображение десятичного действительного числа, программа должна выводить сообщение о номере первого ошибочного символа и завершать выполнение.

Примерный результат работы приложения для случая, когда пользователь ввёл недопустимый символ в изображение действительного числа, приведён на Рисунок **2**.

```
//-----------------------------------------------------------------
   Приложение для перевода десятичных действительных чисел
   в систему счисления с основанием p от 2 до 16
//-----------------------------------------------------------------
  Введите основание с. сч. p - число в диапазоне 2..16: 16
//-----------------------------------------------------------------
  Введите действительное число: 123W.78
//-----------------------------------------------------------------
  ошибка ввода исходного числа: позиция номер 4
//-----------------------------------------------------------------
```

Рисунок 2. Ошибка ввода десятичного числа. Десятичное число переводится в 16-ичную систему счисления.

В варианте 29* при вводе значения основания системы счисления p, выходящего за пределы допустимого диапазона (от 2 до 16), приложения должно сообщать об этой ошибке. Примерный результат работы приложения для этого случая приведён на Рисунок **3**.

```
//-----------------------------------------------------------------
   Приложение для перевода десятичных действительных чисел
   в систему счисления с основанием p от 2 до 16
//-----------------------------------------------------------------
  Введите целое число в диапазоне 2..16: 1
//-----------------------------------------------------------------
  Введите действительное число: -154.5
//-----------------------------------------------------------------
  Неверное основание p = 1
//-----------------------------------------------------------------
```

Рисунок 3. Ошибка ввода основания системы счисления. Десятичное число переводится в 16-ичную систему счисления.

Примерный результат работы приложения для варианта 29* в случае, когда пользователь совершил ошибки обоих типов, приведён на Рисунок 4.

```
//-----------------------------------------------------------------
```

```
   Приложение для перевода десятичных действительных чисел
   в систему счисления с основанием p от 2 до 16
//-----------------------------------------------------------------
  Введите целое число в диапазоне 2..16: 20
//-----------------------------------------------------------------
  Введите действительное число: 23R.89
//-----------------------------------------------------------------
//-----------------------------------------------------------------
  ошибка ввода исходного числа: позиция номер 3
//-----------------------------------------------------------------
  Неверное основание p = 20
//-----------------------------------------------------------------
```

Рисунок 4. Ошибка ввода десятичного числа и основания системы счисления. Десятичное число переводится в 16-ичную систему счисления.

Спецификация приложения для вариантов с 15 по 28 и 30*

1. Приложение работает в режиме диалога с пользователем. Пользователь вводит основание системы счисления p и действительное число в этой системе счисления с клавиатуры. Приложение формирует, десятичное представление введённого числа, и выводит его на экране монитора.

Примерный результат работы приложения для варианта 30* представлен ниже на Рисунок 5. Пользователь ввёл исходные данные без ошибок.

```
//-----------------------------------------------------------------
   Приложение для перевода p-ичных действительных чисел
   в десятичные для основания p от 2 до 16
//-----------------------------------------------------------------
  Введите основание с. сч. p - число в диапазоне 2..16: 16
-------------------------------------------------------------------
  Введите действительное 16-ичное число: -AF.E
//-----------------------------------------------------------------
  Десятичное представление: -175.8750
//-----------------------------------------------------------------
```

Рисунок 5. Ошибки ввода отсутствуют. 16-ичное число переводится в десятичный эквивалент.

2. В приложении необходимо обеспечить контроль ввода исходных данных. При вводе недопустимого значения для основания системы счисления p приложение должно выводить сообщение об ошибке и завершаться. При вводе недопустимых символов в изображение действительного числа в системе счисления с основанием p, программа должна выводить сообщение о номере первого ошибочного символа и завершать своё выполнение.

В варианте 30* при вводе значения основания системы счисления p, выходящего за пределы допустимого диапазона (от 2 до 16), приложения должно сообщать об этой ошибке. Примерный результат работы приложения для этого случая приведён на Рисунок 6.

```
//-----------------------------------------------------------------
   Приложение для перевода p-ичных действительных чисел
   в десятичные для основания p от 2 до 16
//-----------------------------------------------------------------
  Введите основание с. сч. p - число в диапазоне 2..16: 1
//-----------------------------------------------------------------
  Неверное основание p = 1
//-----------------------------------------------------------------
```

Рисунок 6. Ошибка ввода основания системы счисления. Число представленное в системе счисления с основанием p переводится в десятичную систему счисления.

Примерный результат работы приложения для случая, когда пользователь ввёл недопустимый символ в изображение приведён на Рисунок 7.

```
//-------------------------------------------------------------------
   Приложение для перевода p-ичных действительных чисел
   в десятичные для основания p от 2 до 16
//-------------------------------------------------------------------
  Введите основание с. сч. p - число в диапазоне 2..16: 16
//-------------------------------------------------------------------
  Введите действительное число: 123W.78
/-------------------------------------------------------------------
  Ошибочный символ. Позиция номер: 4
//-------------------------------------------------------------------
```

Рисунок 7. Ошибка ввода 16-ичного числа. 16-ичное число переводится в десятичную систему счисления.

Варианты заданий

1. Варианты заданий приведены в Таблица **1**. Номер вашего варианта задания определяется вашим порядковым номером в списке студентов группового журнала. Вы можете выполнить по вашему выбору вариант (29^* или 30^*) повышенной сложности вместо своего варианта.

Таблица 1. Варианты заданий.

Номер варианта	«входной формат»	«выходной формат»	Основание системы счисления p
1	10	p	2
2	10	p	3
3	10	p	4
4	10	p	5
5	10	p	6
6	10	p	7
7	10	p	8
8	10	p	9
9	10	p	11
10	10	p	12
11	10	p	13
12	10	p	14
13	10	p	15
14	10	p	16
15	p	10	2
16	p	10	3
17	p	10	4
18	p	10	5
19	p	10	6
20	p	10	7
21	p	10	8
22	p	10	9
23	p	10	11
24	p	10	12
25	p	10	13

26	p	10	14
27	p	10	15
28	p	10	16
29*	10	p	2..16
30*	p	10	2..16

***Примечание**. Здесь p означает, что число представлено в системе счисления с основанием p, а 10 – что число представлено в 10-ичной системе счисления. Варианты* 29* и 30* *- варианты повышенной сложности. В них пользователь может выбрать основание системы счисления из заданного диапазона от 2 до 16 во время работы приложения.*

2. Для вариантов с 1 по 14 характеристики форматов следующие:
   - «входной» формат - десятичное действительное число,
   - «выходной» формат - действительное число в системе счисления с основанием p, где p - основание системы счисления, определяемое вариантом задания.

   Входной формат определяется синтаксическими правилами языка C# для написания вещественных констант.

   Выходной формат (формат представления числа в системе счисления с основанием p) состоит из четырёх полей и имеет следующий вид:

   [<знак>]<целое><разделитель><дробь>

   Поле <знак> является необязательным, оно может содержать символ «-» или быть пустым (знак плюс не вводится).

   Поле <целое> является обязательным, оно должно содержать целое число в системе счисления с основанием p без знака.

Поле <разделитель> является необязательным, оно содержит символ «.», отделяющий целую часть числа от его дробной части.

Поле <дробь> является необязательным, оно может содержать цифры системы счисления с основанием p дробной части числа.

Пример 1. Правильное представление 16-ичных чисел в «выходном» формате.

0

0.

-2

2.

-275

A27E.

-0.012C

123F.1

5.34

-B25.34

-0.2

3. Для вариантов с 15 по 28 характеристики форматов следующие:
   - «входной» формат - действительное число в системе счисления с основанием p, где p - основание системы счисления, определяемое вариантом,
   - «выходной» формат - десятичное действительное число.

Входной формат состоит из четырёх полей и имеет следующий вид:

[<знак>]<целое><разделитель><дробь>

Суммарное количество цифр n (n = i + f, здесь i - количество цифр в целой части, f - количество цифр в дробной

части) в полях <целое> и <дробь> должно лежать в диапазоне 1 <= n <= N, где N = 12.

Поле <знак> является необязательным, оно может содержать символ «-» или быть пустым (знак плюс не вводится).

Поле <целое> является обязательным, оно должно содержать целое в системе счисления с основанием p без знака, количество p-ичных цифр в котором должно быть 1 =< i <= N (незначащие нули слева не вводятся, исключая случай, когда ноль в поле единственный).

Поле <разделитель> является необязательным, оно содержит символ «.», отделяющий целую часть числа от его дробной части.

Поле <дробь> является необязательным, оно может содержать цифры системы счисления с основанием p дробной части числа. Количество цифр в нём должно принадлежать диапазону 0 =< f <= N - 1.

Пример 2. Правильное представление 3-ичных чисел во «входном» формате.

0.

0

-2

2

-210020212021

212021012122

-0,01221221211

0,012

120012.012212

12001212201.1

1.2

-102.2101

Пример 3. Неправильное представление 3-ичных чисел во «входном» формате для вариантов с 15 по 28.

-0 знак перед числом 0

, отсутствует <целое> и <дробь>;

-, отсутствует <целое> и <дробь>;

-,121 отсутствует поле <целое>;

1201212121200 недопустимая длина (13 разрядов);

1221202,1201212 недопустимая длина (14 разрядов);

-0,200121202211012 недопустимая длина (15 разрядов).

Порядок выполнения

1. Спроектируйте программу и опишите проект с помощью схемы иерархии логических модулей (смотри рис. 8, рис. 11).
2. Поведение каждого логического модуля опишите с помощью HIPO-диаграммы.
3. Для каждого модуля разработайте блок-схему в соответствии с технологией структурного программирования.
4. Реализуйте проект на языке C# в начиная с модулей нижнего уровня в следующем порядке:
   - реализуйте модули преобразования формы представления числа как методы класса Program и протестируйте их по отдельности;
   - реализуйте модули ввода и проверки ввода как методы класса Program и протестируйте их.

Физические модули реализуйте как методы класса Program. Выберите метод тестирования, разработайте тестовые наборы данных и протестируйте каждую метод.

5. Опишите реализацию программы с помощью схемы иерархии физических модулей.

6. Выберите метод тестирования, разработайте тестовые наборы данных, соберите всё приложение и протестируйте его.

Содержание отчета

1. Вариант задания.
2. Схема иерархии логических модулей.
3. HIPO-диаграмма для каждого модуля.
4. Блок-схема для каждого модуля.
5. Схема иерархии физических модулей.
6. Таблицу соответствия логических и физических модулей.
7. Наборы тестовых данных для тестирования каждого модуля и приложения в целом.
8. Текст программы на C# с комментариями.
9. Литература.

Основные сведения

Технология структурного программирования

Традиционная технология программирования складывалась в условиях, когда основными потребителями программ были научные учреждения, вычислительные ресурсы были ограничены, а проблемы сопровождения по существу неизвестны. Основными критериями качества программы считались ее узко понимаемая эффективность и компактность. Со временем сложность программ возросла настолько, что на их разработку уходили годы труда большого коллектива, а в результате системы появлялись с опозданием и содержали тысячи ошибок.

Кризис программного обеспечения привел к необходимости создания нового способа создания программ, который снижал бы общие затраты на протяжении всего жизненного цикла программы, — от замысла до завершения эксплуатации. Такая технология

появилась в начале 70-х годов и была названа структурным программированием. В его основе лежит сочетание теории программирования и личного опыта высококвалифицированных программистов, а также учет современных требований к программам и промышленного характера их производства.

Главное требование, которому должна удовлетворять программа, — работать в полном соответствии со спецификацией и адекватно реагировать на любые действия пользователя. Кроме этого, программа должна быть выпущена точно к заявленному сроку, и допускать оперативное внесение необходимых изменений и дополнений. Иными словами, современные критерии качества программы — это, прежде всего, надежность, а также возможность точно планировать производство программы и ее сопровождение. Для достижения этих целей программа должна иметь простую структуру, быть хорошо читаемой и легко модифицируемой.

Структурное программирование — это технология создания программ, позволяющая путем соблюдения определенных правил уменьшить время разработки и количество ошибок, а также облегчить возможность модификации программы.

Структурный подход охватывает все стадии разработки проекта: анализ требований, проектирование, программирование и тестирование.

Структурный подход к программированию позволил успешно создавать достаточно крупные проекты, но сложность программного обеспечения продолжала возрастать, и требовались все более развитые средства ее преодоления. Идеи структурного программирования получили свое дальнейшее развитие в объектно-

ориентированном программировании (ООП) — технологии, позволяющей достичь простоты структуры и управляемости очень крупных программных систем.

Структурные методы анализа и проектирования ПО

Структурные методы являются строгой дисциплиной системного анализа и проектирования. Методы структурного анализа и проектирования стремятся преодолеть сложность больших систем путем расчленения их на части («черные ящики») и иерархической организации этих «черных ящиков». Выгода в использовании «черных ящиков» заключается в том, что их пользователю не требуется знать, как они работают, необходимо знать лишь их входы и выходы, а также назначение (т.е. функции, которые они выполняет).

Таким образом, первым шагом упрощения сложной системы является ее разбиение на «черные ящики», при этом такое разбиение должно удовлетворять следующим критериям:

- каждый «черный ящик» должен реализовывать единственную функцию системы;
- функция каждого «черного ящика» должна быть легко понимаема независимо от сложности ее реализации;
- связь между «черными ящиками» должна вводиться только при наличии связи между соответствующими функциями системы (например, в бухгалтерии один «черный ящик» необходим для расчета общей заработной платы служащего, а другой для расчета налогов — необходима связь между этими «черными ящиками»: размер заработной платы требуется для расчета налогов);

- связи между «черными ящиками» должны быть простыми, насколько это возможно, для обеспечения независимости между ними.

Второй важной идеей, лежащей в основе структурных методов, является идея иерархии. Для понимания сложной системы недостаточно разбиения ее на части, необходимо эти части организовать определенным образом, а именно в виде иерархических структур. Все сложные системы Вселенной организованы в иерархии: от галактик до элементарных частиц. Человек при создании сложных систем также подражает природе. Любая организация имеет директора, заместителей по направлениям, иерархию руководителей подразделений, рядовых служащих.

Кроме того, структурные методы широко используют визуальное моделирование, служащее для облегчения понимания сложных систем.

Структурным анализом принято называть метод исследования системы, начинающий с ее общего обзора, который затем детализируется, приобретая иерархическую структуру со все большим числом уровней.

Для таких методов характерно:

- разбиение системы на уровни абстракции с ограничением числа элементов на каждом из уровней (обычно от 3 до 6-7);
- ограниченный контекст, включающий лишь существенные на каждом уровне детали;
- использование строгих формальных правил записи;
- последовательное приближение к конечному результату.

В структурном анализе основным методом разбиения на уровни абстракции является функциональная декомпозиция, заключающаяся в декомпозиции (разбиении) системы на функциональные подсистемы, которые, в свою очередь, делятся на подфункции, т.е. - на задачи и так далее до конкретных процедур. При этом система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы «снизу вверх» от отдельных задач ко всей системе целостность теряется, возникают проблемы при описании информационного взаимодействия отдельных компонентов.

Все наиболее распространенные методы структурного подхода базируются на ряде общих принципов. Базовыми принципами являются:

- принцип «разделяй и властвуй» — принцип решения трудных проблем путем разбиения их на множество меньших независимых задач, легких для понимания и решения;
- принцип иерархического упорядочения — принцип организации составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к нежелательным последствиям (вплоть до неудачного завершения проекта). Основными из этих принципов являются:

- принцип абстрагирования — выделение существенных аспектов системы и отвлечение от несущественных;
- принцип непротиворечивости — обоснованность и согласованность элементов системы;

- принцип структурирования данных — данные должны быть структурированы и иерархически организованы.

В структурном анализе и проектировании используются различные модели, описывающие:

- функциональную структуру системы;
- последовательность выполняемых действий;
- передачу информации между функциональными процессами;
- отношения между данными.

Наиболее распространенными моделями первых трех групп являются:

- функциональная модель SADT (Structured Analysis and Design Technique);
- модель IDEF3;
- DFD (Data Flow Diagrams) — диаграммы потоков данных.

Модель «сущность - связь» (ERM — Entity-Relationship Model), описывающая отношения между данными, традиционно используется в структурном анализе и проектировании, однако, по существу, представляет собой подмножество объектной модели предметной области.

Модульное программирование

Прикладные программы чаще всего слишком сложны, чтобы быть написанными как единое целое или разработанными одним программистом. Поэтому сложность программ долгое время вызывала трудности при проектировании систем и программного обеспечения. Программы стали слишком велики, чтобы их можно было представить во всех подробностях как единое целое. Поэтому

большие программы на этапе проектирования стали разбивать на логические (функциональные) модули.

Под модульным программированием (модуляризацией) понимается разделение программы на части по некоторым установленным правилам.

Эти части на этапе кодирования могут быть реализованы в виде модулей или подпрограмм в зависимости от их размера и выбранного языка программирования.

Для того чтобы логический модуль можно было закодировать, необходимо предварительно описать его поведение, т.е. специфицировать. Для спецификации функций логических модулей с точки зрения входных выходных переменных и связи между ними используют HIPO-диаграмму. Её название образовано из первых букв английского словосочетания hierarchy-input-processing-output (иерархическое описание вход-обработка-выход).

Программы разбиваются на модули для того, чтобы:

- упростить их разработку и реализацию;
- облегчить чтение программ;
- упростить их настройку и модификацию;
- облегчить работу с данными, имеющими сложную структуру;
- избежать чрезмерной детализации алгоритмов;
- обеспечить более выгодное размещение программ в памяти ЭВМ.

Правильная декомпозиция является главным способом преодоления сложности разработки больших систем ПО. Понятие «правильная» по отношению к декомпозиции означает следующее:

- количество связей между отдельными модулями должно быть минимальным (принцип «слабой связанности» - Low Coupling);
- связность отдельных частей внутри каждого модуля должна быть максимальной (принцип «сильного сцепления» — High Cohesion).

Связность модуля определяется как мера зависимости его частей.

Чем выше связность модуля, тем лучше результат проектирования. Для обозначения связности используется также понятие силы связности модуля.

Сцепление модулей представляет собой меру относительной независимости модулей, которая определяет их читабельность и сохранность.

Независимые модули могут быть модифицированы без переделки каких-либо других модулей. Слабое сцепление более желательно, так как это означает высокий уровень их независимости. Модули являются полностью независимыми, если каждый из них не содержит о другом никакой информации. Чем больше информации о других модулях используется в них, тем менее они независимы и тем теснее сцеплены.

Методы проектирования программ, основанные на модульном принципе, делятся на три группы:

- методы нисходящего проектирования,
- методы расширения ядра и
- методы восходящего проектирования.

На практике обычно применяются различные сочетания этих методов.

HIPO - диаграмма

HIPO - диаграмма предназначена для спецификации (описания) требований к программе или функциональному (логическому) модулю.

Она является результатом анализа процесса обработки данных, выполняемого программой, с точки зрения исходных данных и результатов работы программы.

HIPO - диаграмма имеет следующий вид:

В разделе “вход” перечисляются имена входных переменных, их типы, диапазоны возможных значений.

В разделе “выход” перечисляются имена выходных переменных, их типы, диапазоны возможных значений.

В разделе “обработка” для каждой выходной переменной необходимо указать, с какими входными переменными она связана и как. В этом разделе содержится описание того, что делает программа, а не как она это делает.

Таким образом, HIPO-диаграмма – это описание поведения процесса обработки данных в таких существенных признаках, как входные значения, выходные значения и связь между ними.

Метод нисходящего проектирования

Метод нисходящего проектирования подобен методу получения детального изображения из более общего вида с

помощью телескопического увеличения. На, начальном шаге формируется предложение, описывающее функцию всей программы. Затем определяются ее подфункции. Эта процедура является рекурсивной, т. е., следуя ей, каждая из подфункций может расчленяться до тех пор, пока ее составные части не будут окончательно уточнены. Метод нисходящего проектирования, иногда называемый функциональной декомпозицией, основан на двух стратегиях: пошаговом уточнении, разработанном Е. Дейкстрой, и анализе сообщений, базирующемся на работах Йодана, Константайна и Мейерса. Эти стратегии отличаются способами определения начальных спецификаций, методами, используемыми при разбиении задачи на части, и правилами записи.

Метод расширения ядра

Метод расширения ядра отличается от способа нисходящего проектирования: в нем больше внимания вначале уделяется выявлению множества вспомогательных функций, а не определению функции всей программы в целом. Эти функции можно получить, применяя методы проектирования структур данных, которые используются при иерархическом модульном проектировании, разработанном Джексоном, или определяя области хранения данных с последующим анализом связанных с ними функциональных единиц (как в методе определения спецификаций модуля, разработанном Парнасом)

Метод восходящего проектирования

При использовании метода восходящего проектирования в первую очередь определяются вспомогательные функции, которые могут потребоваться для проектируемой программы. Модульная декомпозиция заключается в нахождении ключевых модулей

промежуточных уровней, которые затем разрабатываются восходящим и нисходящим способами одновременно. Эти модули не являются вспомогательными в том смысле, что потребность в них возникает в нескольких точках программы. Необходимость в использовании этих модулей может возникать в других программах или системах.

Функции, определяемые как вспомогательные при восходящем проектировании, реализуются с помощью модулей самых нижних уровней.

Базовые управляющие структуры структурного программирования

В теории программирования доказано, что программу для решения задачи любой сложности можно составить только из трех структур, называемых следованием, ветвлением и циклом. Этот результат установлен Боймом и Якопини еще в 1966 году путем доказательства того, что любую программу можно преобразовать в эквивалентную, состоящую только из этих структур и их комбинаций.

Следование, ветвление и цикл называют базовыми конструкциями структурного программирования. Следованием называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных). Ветвление задает выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия. Цикл задает многократное выполнение оператора Рисунок 9. Особенностью базовых конструкций является то, что любая из них имеет только один вход и один выход, поэтому конструкции могут вкладываться друг в друга произвольным образом, например, цикл может

содержать следование из двух ветвлений, каждое из которых включает вложенные циклы Рисунок 10.

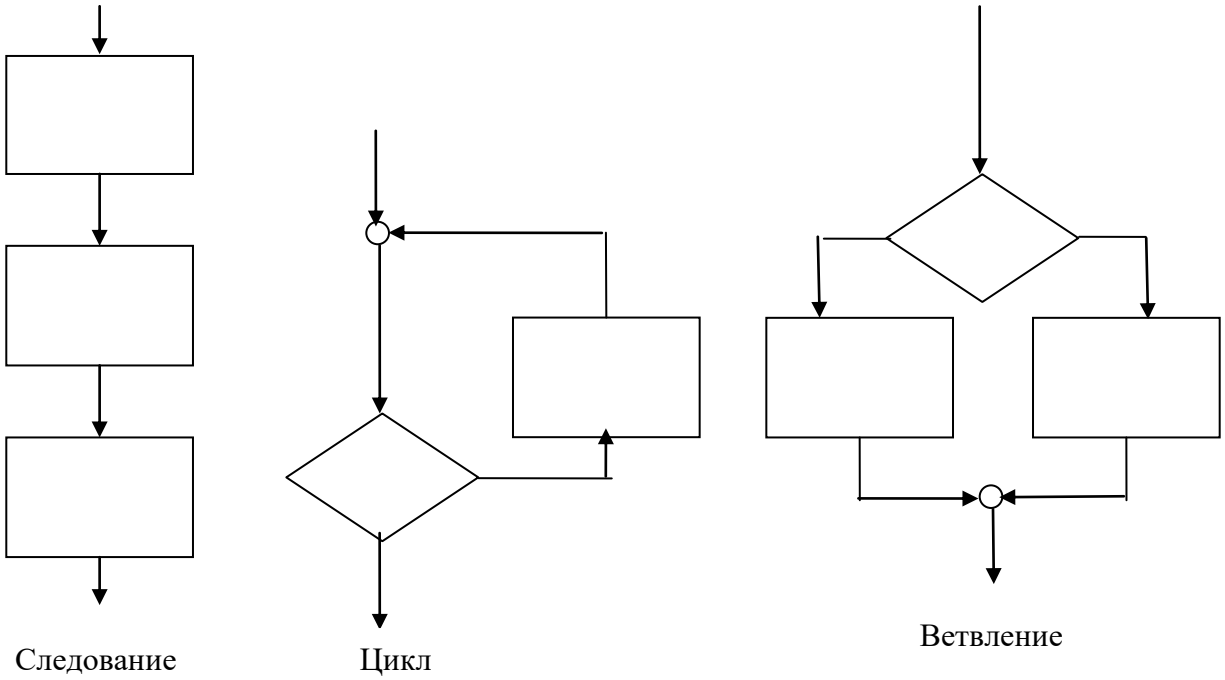

Рисунок 9. Базовые конструкции структурного программирования.

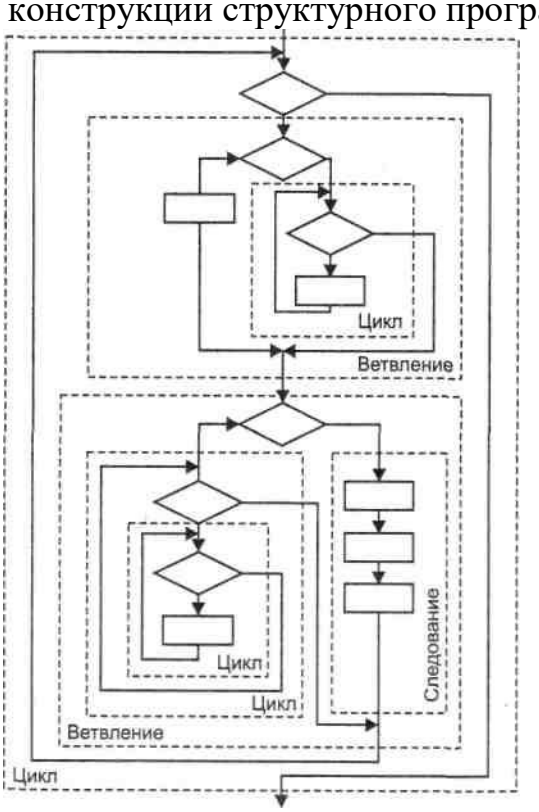

Рисунок 10. Вложение базовых конструкций

Целью использования базовых конструкций является получение программы простой структуры. Такую программу легко

читать (а программы чаще приходится читать, чем писать), отлаживать и при необходимости вносить в нее изменения. Структурное программирование часто называли «программированием без goto», и в этом есть большая доля правды: частое использование операторов передачи управления в произвольные точки программы затрудняет прослеживание логики ее работы. С другой стороны, никакие принципы нельзя возводить в абсолют, и есть ситуации, в которых использование goto оправдано и приводит, напротив, к упрощению структуры программы.

В большинстве языков высокого уровня существует несколько реализаций базовых конструкций. Они введены для удобства программирования, и в каждом случае надо выбирать наиболее подходящие средства. Главное, о чем нужно помнить даже при написании самых простых программ, — что они должны состоять из четкой последовательности блоков строго определенной конфигурации.

Проектирование и тестирование программы

Структурный подход к программированию, как уже упоминалось, охватывает все стадии разработки проекта: анализ требований, проектирование, собственно программирование (кодирование) и тестирование. Задачи, которые при этом ставятся, — уменьшение числа возможных ошибок за счет применения только допустимых структур, возможно более раннее обнаружение ошибок и упрощение процесса их исправления. Ключевыми идеями структурного подхода являются

- нисходящая разработка,
- структурное программирование и
- нисходящее тестирование.

Приведенные ниже этапы создания программ рассчитаны на достаточно большие проекты, разрабатываемые коллективом программистов. Для программы небольшого объема каждый этап упрощается, но содержание и последовательность этапов не изменяются.

I этап. Создание любой программы начинается с анализа требований пользователей - постановки задачи. Изначально задача ставится в терминах предметной области, и необходимо перевести ее в термины, более близкие к программированию. Поскольку программист редко досконально разбирается в предметной области, а заказчик — в программировании (простой пример: требуется написать бухгалтерскую программу), постановка задачи может стать весьма непростым итерационным процессом. Кроме того, при постановке задачи заказчик зачастую не может четко и полно сформулировать свои требования и критерии.

Постановка задачи завершается созданием технического задания, а затем внешней спецификации программы, включающей в себя:

- описание исходных данных и результатов (типы, форматы, точность, способ передачи, ограничения);
- описание задачи, реализуемой программой;
- способ обращения к программе;
- описание возможных аварийных ситуаций и ошибок пользователя.

Таким образом, программа рассматривается как черный ящик, для которого определена функция и входные и выходные данные.

II этап. Разработка внутренних структур данных. Большинство алгоритмов зависит от того, каким образом

организованы данные, поэтому интуитивно ясно, что начинать проектирование программы надо не с алгоритмов, а с разработки структур, необходимых для представления входных, выходных и промежуточных данных. При этом принимаются во внимание многие факторы, например, ограничения на размер данных, необходимая точность, требования к быстродействию программы. Структуры данных могут быть статическими или динамическими.

III этап. Проектирование (определение общей структуры и взаимодействия модулей). На этом этапе применяется технология нисходящего проектирования программы, основная идея которого теоретически проста: разбиение задачи на подзадачи меньшей сложности - функциональные (логические) модули, которые можно рассматривать раздельно. Главный критерий разбиения (функциональной декомпозиции) — минимизация взаимодействия модулей. При этом используется метод пошаговой детализации. Можно представить себе этот процесс так, что сначала программа пишется на языке некоторой гипотетической машины, которая способна понимать самые обобщенные действия, а затем каждое из них описывается на более низком уровне абстракции, и так далее. Очень важной на этом этапе является спецификация интерфейсов, то есть способов взаимодействия подзадач.

Для каждой подзадачи составляется внешняя спецификация, аналогичная приведенной выше. Для каждого подзадачи (модуля) описывается его функция, например, с помощью HIPO-диаграммы. Одна задача может реализовываться с помощью нескольких модулей и, наоборот, в одном модуле может решаться несколько задач. На более низкий уровень проектирования переходят только после окончания проектирования верхнего уровня. Для каждого модуля разрабатывается алгоритм обработки данных с помощью базовых

конструкций структурного программирования и записывается в обобщенной форме — например, словесной, в виде обобщенных блок-схем или другими способами.

На этапе проектирования следует учитывать возможность будущих модификаций программы и стремиться проектировать программу таким образом, чтобы вносить изменения было возможно проще.

Процесс проектирования является итерационным, поскольку в программах реального размера невозможно продумать все детали с первого раза.

IV этап. Структурное программирование. Процесс программирования также организуется по принципу «сверху вниз»: вначале кодируются модули самого верхнего уровня и составляются тестовые примеры для их отладки, при этом на месте еще не написанных модулей следующего уровня ставятся «заглушки» — временные программы. «Заглушки» в простейшем случае просто выдают сообщение о том, что им передано управление, а затем возвращают его в вызывающий модуль. В других случаях «заглушка» может выдавать значения, заданные заранее или вычисленные по упрощенному алгоритму.

Таким образом, сначала создается логический скелет программы, который затем обрастает плотью кода.

Можно применять к процессу программирования восходящую технологию — написать и отладить сначала модули нижнего уровня, а затем объединять их в более крупные фрагменты, но этот подход имеет ряд недостатков.

Во-первых, в процессе кодирования верхнего уровня могут быть вскрыты те или иные трудности проектирования более низких уровней программы (просто потому, что при написании программы ее логика продумывается более тщательно, чем при проектировании). Если подобная ошибка обнаруживается в последнюю очередь, требуются дополнительные затраты на переделку уже готовых модулей нижнего уровня.

Во-вторых, для отладки каждого модуля, а затем более крупных фрагментов программы требуется каждый раз составлять свои тестовые примеры, и программист часто вынужден имитировать то окружение, в котором должен работать модуль. Нисходящая же технология программирования обеспечивает естественный порядок создания тестов — возможность нисходящей отладки, которая рассмотрена далее.

Этапы проектирования и программирования совмещены во времени: в идеале сначала проектируется и кодируется верхний уровень, затем — следующий, и так далее. Такая стратегия применяется потому, что в процессе кодирования может возникнуть необходимость внести изменения, отражающиеся на модулях нижнего уровня.

V этап. Нисходящее тестирование. Этот этап записан последним, но это не значит, что тестирование не должно проводиться на предыдущих этапах. Проектирование и программирование обязательно должны сопровождаться написанием набора тестов — проверочных исходных данных и соответствующих им наборов ожидаемых результатов.

Необходимо различать процессы тестирования и отладки программы.

Тестирование — это процесс выполнения программ с целью обнаружения ошибок.

Ошибкой считается любое не соответствие работы программы заданной на неё спецификации.

Хорошим считается тест, который имеет высокую вероятность обнаружения еще не выявленной ошибки. Удачным считается тест, который обнаруживает еще не выявленную ошибку.

Отладка программы - это процесс, осуществляемый после удачного выполнения теста. Процесс отладки начинается при обнаружении ошибки (например, при удачном завершении теста) и проводится в два этапа:

- определяется природа и местонахождение подозреваемой ошибки в программе;
- фиксируется или исправляется ошибка.

Отладка — процесс исправления ошибок в программе, при этом цель исправить все ошибки не ставится. Исправляют ошибки, обнаруженные при тестировании. При планировании следует учитывать, что процесс обнаружения ошибок подчиняется закону насыщения, то есть большинство ошибок обнаруживается на ранних стадиях тестирования, и чем меньше в программе осталось ошибок, тем дольше искать каждую из них.

Для исчерпывающего тестирования программы необходимо проверить каждую из ветвей алгоритма. Общее число ветвей определяется комбинацией всех альтернатив на каждом этапе. Это конечное число, но оно может быть очень большим, поэтому программа разбивается на фрагменты, после исчерпывающего тестирования, которых они рассматриваются как элементарные узлы

более длинных ветвей. Кроме данных, обеспечивающих выполнение операторов в требуемой последовательности, тесты должны содержать проверку граничных условий (например, переход по условию x> 10 должен проверяться для значений, больших, меньших и равных 10). Отдельно проверяется реакция программы на ошибочные исходные данные.

Идея нисходящего тестирования предполагает, что к тестированию программы приступают еще до того, как завершено ее проектирование. Это позволяет раньше опробовать основные межмодульные интерфейсы, а также убедиться в том, что программа в основном удовлетворяет требованиям пользователя. Только после того как логическое ядро испытано настолько, что появляется уверенность в правильности реализации основных интерфейсов, приступают к кодированию и тестированию следующего уровня программы.

Методы

Методы определяют поведение класса. *Метод* — это *функциональный элемент* класса. Он реализует вычисления или другие действия, выполняемые классом или экземпляром.

Метод представляет собой законченный фрагмент кода, к которому можно обратиться *по* имени. Он описывается один раз, а вызываться может столько раз, сколько необходимо. Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

Синтаксис метода:

```
[ атрибуты ] [ спецификаторы ] тип имя_метода ( [ параметры ] )
        тело_метода
```

Первая часть описания представляет собой *заголовок* метода или сигнатуру. *Тело метода* задаёт действия, выполняемые методом. Оно чаще всего представляет собой блок.

Методы со *спецификатором* доступа public составляют *интерфейс* класса — то, с чем работает *пользователь*. Методы со спецификатором static не работают с экземплярами класса.

Пример простейшего метода:

```
// Метод возвращает значение числа пи.
public static double Pi()
{
    return 3.1415;
}
```

Тип определяет, *значение* какого типа вычисляется с помощью метода. Часто употребляется термин "метод возвращает *значение*". Если метод не возвращает никакого значения, в его заголовке задается тип void, а оператор return отсутствует.

С помощью *параметров* метод обменивается данными с вызывающим его методом. *Параметр* представляет собой локальную переменную, которая при вызове метода принимает *значение* соответствующего фактического параметра (параметра переданного при вызове). *Область действия* параметра — весь метод.

Например, чтобы выделить целую часть вещественной величины a, мы передаем ее в качестве аргумента в метод `Truncate` класса Math, а чтобы вывести *значение* переменной r на экран, мы передаем ее в метод WriteLine класса Console:

```
double a = 2.56;
//Вызов метода в составе выражения.
double r = Math.Truncate(a);
//Вызов метода отдельным оператором.
Console.WriteLine(r);
```

При этом метод `Truncate` возвращает в точку своего вызова вещественное *значение* - целую часть a, которое присваивается переменной r, а метод WriteLine ничего не возвращает. Иллюстрация вызова метода приведена на рис. 10.

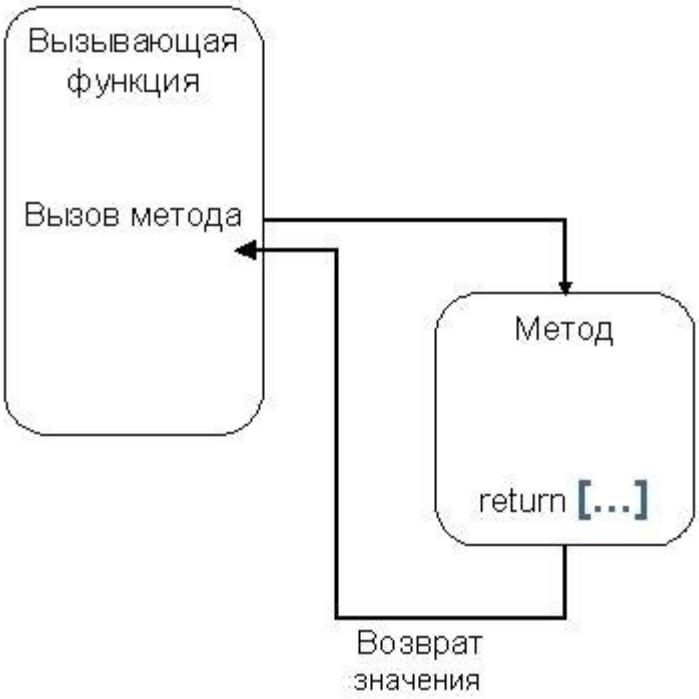

Рис. 10. Вызов метода

Внимание

Метод, не возвращающий значение, вызывается отдельным оператором, а метод, возвращающий значение, — в составе выражения в правой части оператора присваивания.

Параметры, описываемые в заголовке метода, определяют множество значений *аргументов*, которые можно передавать в метод. *Список* аргументов при вызове как бы накладывается на *список* параметров, поэтому они должны попарно соответствовать друг другу. Для каждого параметра должны задаваться его *тип* и *имя*. Например, *заголовок метода* Sin выглядит следующим образом:

```
public static double Sin( double a );
```

Имя метода вместе с количеством, типами и спецификаторами его параметров представляет собой *сигнатуру метода*. В классе не должно быть методов с одинаковыми сигнатурами.

В примере ниже в класс Program добавлен метод `Fact` с рекурсивным вызовом.

```
using System;

namespace Example_1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(Fact(5));
        }
        //Вычисляет факториал n.
        static int Fact(int n)
        {
            int res = 1;
            if (n > 1) res = n * Fact(n - 1);
            return res;
        }
    }
}
```

Результат вызова метода:

```
120
Для продолжения нажмите любую клавишу . . .
```

Параметры методов

При вызове метода выполняются следующие действия:

1. Вычисляются выражения, стоящие на месте аргументов.
2. Выделяется память под параметры метода в соответствии с их типом.

3. Каждому из параметров сопоставляется соответствующий аргумент (аргументы как бы накладываются на параметры и замещают их).

4. Выполняется тело метода.

5. Если метод возвращает значение, оно передается в точку вызова; если метод имеет тип void, управление передается на оператор, следующий после вызова.

При этом проверяется соответствие типов аргументов и параметров и при необходимости выполняется их преобразование. При несоответствии типов выдается *диагностическое сообщение*. Пример ниже иллюстрирует этот процесс.

```
using System;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
            // Вызов метода Min.
            Console.WriteLine( Min(5,9) );
            // Результат: 5.
            int a = 2, b = 4;
            // Вызов метода Min.
            int r = Min( a, b );
            // Результат: 2.
            Console.WriteLine( r );

            short c = 3, d = 4;
            // вызов метода Min.
            int w = Min(c, d);
            // Результат: 3.
            Console.WriteLine( w );
            // вызов метода Min.
            int v = Min(a + c, d / 2 * b);
            // Результат: 5.
            Console.WriteLine( v );
        }
           // Метод выбора минимального значения.
        static int Min(int c, int d)
        {
```

```
            if ( c < d ) return c;
            else         return d;
        }
    }
}
```

Внимание

Главное требование при передаче параметров состоит в том, что аргументы при вызове метода должны записываться в том же порядке, что и в заголовке метода, и должно существовать неявное преобразование типа каждого аргумента к типу соответствующего параметра. Количество аргументов должно соответствовать количеству параметров. Иллюстрация приведена на рис. 11.

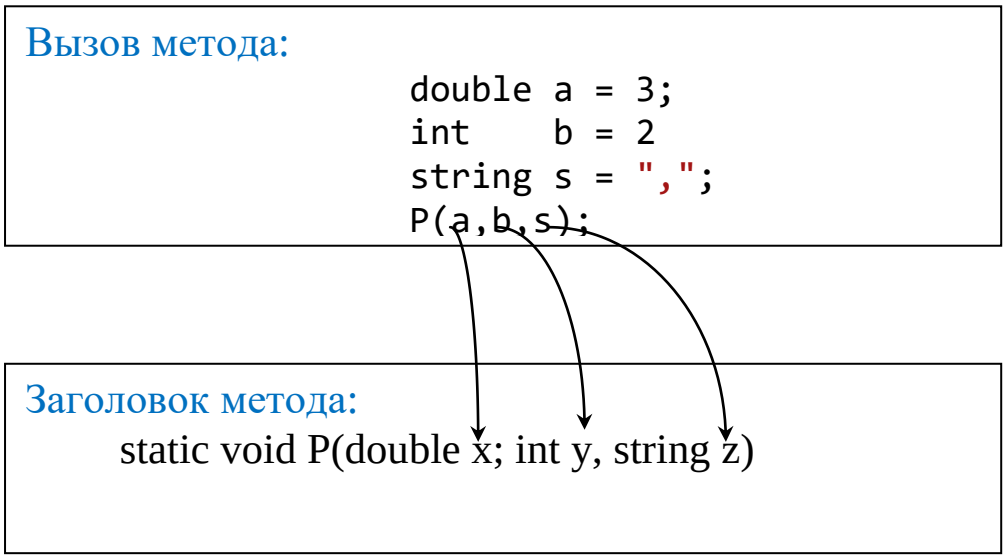

Рис. 11. Соответствие параметров при вызове метода

Существуют два способа передачи параметров: *по* значению и *по* ссылке.

При передаче по значению метод получает копии значений аргументов, и *операторы* метода работают с этими копиями. Доступа к исходным значениям аргументов у метода нет, а, следовательно, нет и возможности их изменить.

При передаче по ссылке (*по адресу*) метод получает копии *адресов аргументов*, он осуществляет *доступ* к ячейкам памяти *по* этим адресам и может изменять исходные значения аргументов, модифицируя параметры.

В C# для обмена данными между вызывающей и вызываемой функциями предусмотрено четыре типа параметров:

- параметры-значения;
- параметры-ссылки — описываются с помощью ключевого слова ref ;
- выходные параметры — описываются с помощью ключевого слова out ;
- параметры-массивы — описываются с помощью ключевого слова params.

Ключевое слово предшествует описанию типа параметра. Если оно опущено, *параметр* считается параметром-значением. *Параметр*-массив может быть только один и должен располагаться последним в списке, например:

```
public int Calculate( int a, ref int b, out int c, params int[] d ) …
```

Пример выполнения. Вариант задания 29*. Варианты с 1 по 14 выполняются по аналогии.

Перевод десятичного действительного числа со знаком целой и дробной частью в эквивалентное ему число, представленное в системе счисления с основанием p (p-ичное число), представленное строкой. P принадлежит диапазону значений 2..15.

Схема иерархии логических модулей.

Ниже на Рисунок 11 отображён результат функциональной декомпозиции приложения.

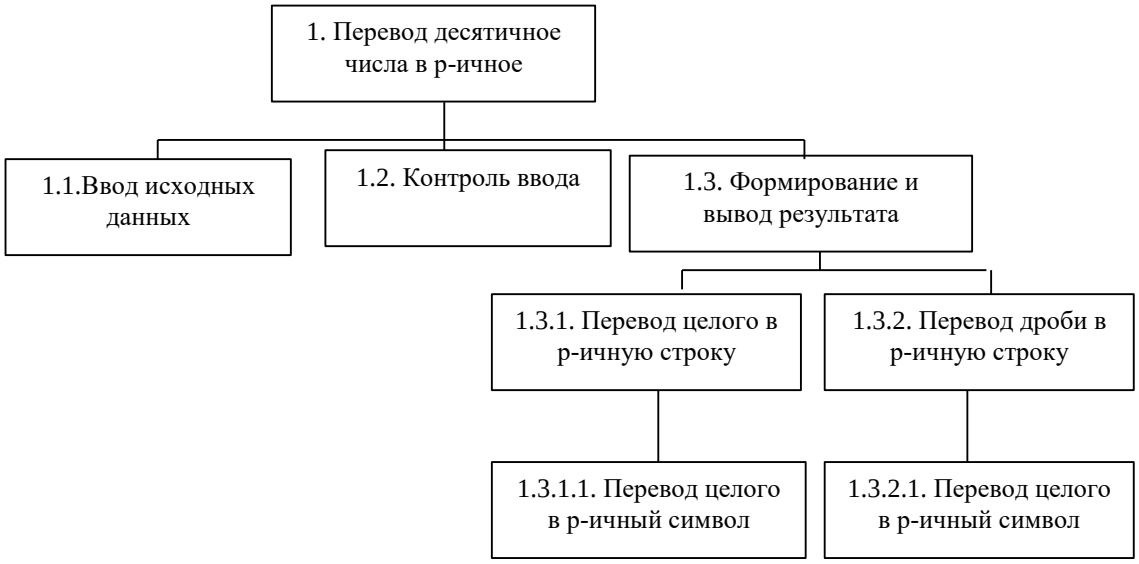

Рисунок 11. Схема иерархии логических модулей. Вариант задания 29*.

Каждая функция, выполняемая приложением, отображена соответствующим логическим модулем на схеме иерархии. Логические модули на схеме иерархии представлены прямоугольниками. Прямые линии указывают порядок передачи управления между модулями.

HIPO-диаграммы для логических модулей

1. Модуль «1. Перевод десятичного числа в р-ичное число, представленное строкой».

Разделы «Вход» и «Выход» пустые, потому что этот модуль не имеет формальных параметров. Он взаимодействует с пользователем через клавиатуру и монитор.

Вход	Обработка	Выход
	Пользователь вводит с клавиатуры терминала десятичное действительное число N и целое число p (основание системы счисления) в диапазоне от 2 до 16. Приложение вычисляет и выводит на монитор результат – строку Np, содержащую p-ичное представление числа N. Если при вводе исходных данных пользователь допустил ошибку, приложение выводит на монитор сообщение об ошибке.	

2. Модуль «1.1. Ввод исходных данных».

Вход	Обработка	Выход
	Пользователь вводит с клавиатуры терминала действительное десятичное число в переменную N и целое число в переменную p – основание системы счисления в диапазоне от 2 до 16.	N – вещественная переменная; p – целая переменная.

3. Модуль «1.2. Контроль ввода».

Вход	Обработка	Выход
p – целое число.	Если значение переменной p (основание системы счисления), находится вне диапазона от 2 до 16, в строку S заносим сообщение об ошибке ввода – «неверная система счисления», и возвращаем значение true. В противном случае – строка S получает значение пустой строки, а возвращаем значение false.	S – строка; булевское значение

4. Модуль «1.3. Формирование результата».

Вход	Обработка	Выход
N – вещественное значение; p – целое значение в диапазона от 2 до 16.	Строка Np содержит представление числа N в системе счисления с основанием p.	Np - строка.

5. Модуль «1.3.1. Перевод целого в p-ичную строку».

Вход	Обработка	Выход
I – целое неотрицательное число; p – целое число в диапазоне 2..16.	Ip – строка, содержащая p-ичное представление числа I. p – основание системы счисления.	Ip – строка

6. Модуль «1.3.2. Перевод дроби в p-ичную строку».

Вход	Обработка	Выход
F – неотрицательная дробь; p – целое число в диапазоне 2..16.	Fp – строка, содержащая p-ичное представление дроби F. p – основание системы счисления.	Fp – строка.

7. Модуль «1.3.1.1. Перевод целого в p-ичный символ».

Вход	Обработка	Выход
d – целое число, соответствующее p-ичной цифре.	ch – символ, соответствующий p- ичной цифре d. p – основание системы счисления в диапазоне 2..16.	ch – символ, соответствующий p-ичной цифре.

Пример структуры консольного приложения для перевода десятичного действительного числа в p-ичное представление представленное строкой.

Соответствие между логическими и физическими модулями для представлено в таблице 2.

```
using System;

namespace Rgz10ToP
{
    class Program
    {
        //Перевод десятичного числа в p-ичное представление
        //Точность представления p-ичного числа:
        //число разрядов в дробной части
        const int acc = 20;
        static void Main(string[] args)
        {
            // Основание системы счисления.
```

```
            int p;
            //Строка представляющая введённое 10-ичное число.
            string Np;
            //Исходное 10-тичное число.
            double N;
            //Позиция ошибки во введённом 10-ичном числе.
            int mistake;
            //Сообщение об ошибочном значении p.
            string s;
            Input(out Np,out p);
            N = double.Parse(Np);
            Console.WriteLine(FormResult(N, p));
        }
        static string IntToPChar(int d)
        //Переводит цифру в её строковое представление
        //в системах счислени с основаниями от 2 до 16.
        {
        }
        static string Int10ToP(int r, int p)
        //Перевод целой части в строку, содержащую её p-ичное
        //представление.
        {
        }
        static string Frac10ToP(double f, int p)
        //Перевод дробной части в строку, содержащую её
        //p-ичное
        //представление.
        {
        }
        static void Input(out string Np, out int p)
        //Ввод 10-тичного числа в строку и основания системы
         //счисления p.
        {
        }
        static bool Check_p(int p, out string s)
        //Контроль ввода.
        {
        }
        static string FormResult(double N, int p)
        //Формирование результата
        {
        }
    }
}
```

Таблица 2. Соответствие между логическими и физическими модулями для вариантов с 1 по 14 и 29*.

Номер	Логический модуль	Физический модуль
1.	Перевод десятичного числа в р-ичное число, представленное строкой	`static void Main(string[] args)`
1.1.	Ввод исходных данных.	`static void Input(out string Np, out int p)`
1.2.	Контроль ввода.	`static bool Check_p(int p, out string s)`
1.3.	Формирование результата.	`static string FormResult(double N, int p)`
1.3.1.	Перевод целого в р-ичную строку.	`static string Int10ToP(int r, int p)`
1.3.2.	Перевод дроби в строку, изображающую р-ичное число.	`static string Frac10ToP(double f, int p)`
1.3.1.1. 1.3.2.1.	Перевод целого в р-ичный символ.	`static string IntToPChar(int d)`

Пример выполнения. Вариант задания 30*. Варианты с 15 по 28 выполняются по аналогии

Перевод действительного числа со знаком целой и дробной частями представленного в системе счисления с основанием р (р-ичное число) в эквивалентное ему число десятичное число. Р принадлежит диапазону значений 2..15.

Схема иерархии логических модулей.

Ниже на рис. 11 отображён результат функциональной декомпозиции приложения.

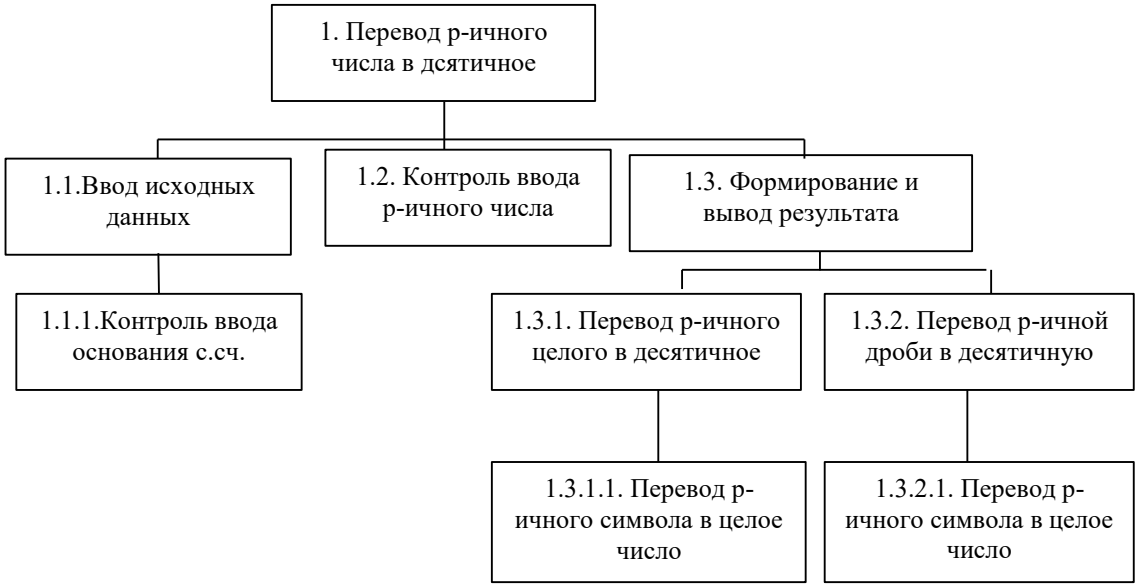

Рисунок 12. Схема иерархии логических модулей. Вариант задания 30*.

Каждая функция, выполняемая приложением, отображена соответствующим логическим модулем на схеме иерархии. Логические модули на схеме иерархии представлены прямоугольниками. Прямые линии указывают порядок передачи управления между модулями.

HIPO-диаграммы для логических модулей

1. Модуль «1. Перевод р-ичного числа в десятичное».

Разделы «Вход» и «Выход» пустые, потому что этот модуль не имеет формальных параметров. Он взаимодействует с пользователем через клавиатуру и монитор.

Вход	Обработка	Выход
	Пользователь вводит с клавиатуры терминала целое число р (основание системы счисления) в диапазоне от 2 до 16 и строку Np изображающую действительное число с целой и	

	дробной частями в системе счисления с основанием p. Приложение вычисляет и выводит на монитор результат – строку N, содержащую десятичное представление числа Np. Если при вводе исходных данных пользователь допустил ошибку, на монитор выводится сообщение об ошибке ввода исходных данных.	

2. Модуль «1.1. Ввод исходных данных».

Вход	Обработка	Выход
	Пользователь вводит с клавиатуры терминала целое число p – основание системы счисления (в диапазоне от 2 до 16). Если p введено без ошибок, то пользователь вводит строку Np, содержащую представление действительного числа в системе счисления с основанием p, в противном случае выводится сообщение об ошибке и приложение завершается.	p – целая переменная; Np – строка.

3. Модуль «1.2. Контроль ввода основания с.сч.».

Вход	Обработка	Выход
p – целое число; Np – строка, содержащая действительное число в системе счисления с основанием p.	Если значение переменной p (основание системы счисления), находится вне диапазона от 2 до 16, в строку s заносим сообщение об ошибке ввода – «неверная система счисления», и возвращаем значение true. В противном случае – строка s получает значение пустой строки, и возвращаем значение false.	s – строка; булевское значение

4. Модуль «1.1.1. Контроль ввода p-ичного числа».

Вход	Обработка	Выход
p – целое число; Np – строка, содержащая действительное число в системе счисления с основанием p.	Если в строке Np нет недопустимых символов, возвращаем значение False. В противном случае в строку s заносим сообщение о номере позиции ошибочного символа и возвращаем значение True.	s – строка; булевское значение

5. Модуль «1.3. Формирование результата».

Вход	Обработка	Выход
Np - строка. p – целое число: основание системы счисления в диапазоне от 2 до 16.	N числовое представление строки Np.	N – вещественное число.

Модуль «1.3.1. Перевод целого в р-ичную строку».

Вход	Обработка	Выход
Ip – строка, содержащая целое неотрицательное число в системе счисления с основанием p; p – целое число в диапазоне 2..16.	I – числовое представление р-ичного неотрицательного числа Ip.	I – неотрицательное целое число

6. Модуль «1.3.2. Перевод дроби в р-ичную строку».

Вход	Обработка	Выход
Fp – строка, содержащая дробное неотрицательное число в системе счисления с основанием p; p – целое число в диапазоне 2..16.	F – числовое представление р-ичной неотрицательной дроби Fp.	Fp – неотрицательное вещественное число.

7. Модуль «1.3.1.1. Перевод целого в р-ичный символ».

Вход	Обработка	Выход
ch – символ, соответствующий р-ичной цифре.	d – целое число, соответствующее – символу(р- ичной цифре). р – основание системы счисления в диапазоне 2..16.	d – целое число, соответствующее р-ичной цифре.

Пример структуры консольного приложения для перевода р-ичного числа в его десятичный эквивалент.

Соответствие между логическими и физическими модулями представлено в таблице 3.

```
using System;

namespace RgzPTo10
{
    class Program
    {
        //Разделитель целой и дробной частей числа.
        const string delim = ".";
        //Перевод  р - ичного числа в десятичное.
        static void Main(string[] args)
        {
            //Позиция ошибки в р-ичном числе.
            int mistake;
            //Исходное р-ичное число.
            string Np;
            //Вспомогательная переменная.
            string z;
            //Основание системы счисления.
            int p;
            Input(out Np, out p);
            Console.WriteLine(FormResult(Np, p));
        }
        static int PCharToInt(char ch)
        //Переводит символ, изображающий цифру, в её числовое
        //представление в системах счисления с основаниями от
         //2 до 16
```

```
        {
        }
        static int PStrToInt(string r, int b)
        //Перевод целое  строка s в её числовое
        //представление в с. сч. с основанием b
        {
        }
        static double PStrToFrac(string f, int b)
        //Перевод дробь строка f в её числовое
        //представление в с. сч. с основанием b
        {
        }
        static bool Check_p(int p, string s)
        //Контроль основания с.сч. p
        {
        }
        static void Input(out string s, out int p)
        //Ввод исходных данных
        {
        }
        static bool Check_number(int p, string Np, out string
s)
        //Контроль p-ичного числа
        {
        }
        static double FormResult(string Np, int p)
        //Формирование результата
        {
        }
    }
}
```

Таблица 3. Соответствие между логическими и физическими модулями для вариантов с 15 по 28 и 30*.

Номер	Логический модуль	Физический модуль
1.	Перевод p-ичного числа в десятичное.	`static void Main(string[] args)`
1.1.	Ввод исходных данных.	`static void Input(out string s, out int p)`
1.1.1.	Контроль введённого основания системы счисления.	`static bool Check_p(int p, string s)`
1.2.	Контроль введённого p-ичного числа.	`static bool Check_number(int p, string Np, out string s)`

1.3.	Формирование результата.	`static double FormResult(string Np, int p)`
1.3.1.	Перевод р-ичного целого в десятичное.	`static int PStrToInt(string r, int b)`
1.3.2.	Перевод р-ичной дроби в десятичную.	`static double PStrToFrac(string f, int b)`
1.3.1.1. 1.3.2.1.	Перевод р-ичного символа в целое число.	`static int PCharToInt(char ch)`

Основная литература

1. Павловская Т.А. C#. Программирование на языке высокого уровня :Учебник для вузов. - СПб. : Питер, 2014. - 432 с. : ил. - (Серия "Учебник для вузов").
2. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.0 на языке C# . 3-е изд.: - СПб.:Питер, 2012. - 928 с. : ил.
3. Справочник по C# [Электронный ресурс] URL: https://msdn.microsoft.com/ru-ru/library/618ayhy6.aspx (дата обращения 19.12.17)
4. METANIT.COM. Сайт о программировании [Электронный ресурс] URL: http://metanit.com/sharp/tutorial/ (дата обращения 19.12.17)