# ОЦЕНКА ХАРАКТЕРИСТИК ПРОГРАММ НА ОСНОВЕ ПРОЦЕДУРНООРИЕНТИРОВАННЫХ МЕТРИК

## ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

### Метрики на основе функциональных указателей

Оценка уровня качества процедурно-ориентированных программных средств с использованием расчетных методов может проводиться на основе применения метрики дефектов качества:

$$DQ = \frac{\text{Количество ошибок (единиц)}}{\text{Количество строк программы (тыс. строк)}}$$

Очевидно, что чем больше значение метрики $DQ$, тем ниже уровень качества программы. При этом предполагается соблюдение нормы $0 \leq DQ \leq 1$.

В практической деятельности количество строк программы часто именуется термином $LOC$ ($Lines\ Of\ Code$). Указанную метрику качества можно рассчитать не на основе регистрации и подсчета ошибок, а на основе так называемых *функциональных указателей* ($FP - Function\ Points$). При этом в качестве показателя рассматривается не количество строк программы, а функциональные указатели. Несмотря на то что функциональные указатели определяются несколько субъективно, все же при этом они характеризуют именно функциональную сложность программы.

Количество функциональных указателей является мерой внешних обращений к данным, под которыми понимают ссылки на внешний ввод, внешний вывод, внешний запрос, автономный локальный файл, разделяемый глобальный файл.

Количество функциональных указателей рассчитывают по формуле

$$FP = F \cdot (0.65 + 0.01 \sum_{i=2}^{14} k_i$$

где $k_i$ – коэффициенты регулировки сложности, зависящие от ответов на вопросы, связанные с особенностями конкретной программы (поскольку таких вопросов 14, именно по такому количеству ведется суммирование значений коэффициентов); $F$ – общее количество функциональных указателей,

определяемое по следующей формуле:

$$F = \sum_{j=1}^{5} f_i.$$

Для расчета количества функциональных указателей используются оценочные элементы, приведенные в выражении как -е элементы:

- количество внешних вводов (вводов данных пользователем) $f_1$
- количество внешних выводов данных (отчеты, экраны, распечатки, сообщения) $f_2$;
- количество внешних запросов (диалоговых вводов-выводов) $f_3$;
- количество локальных внутренних логических файлов (группа логически связанных данных, которая размещена внутри приложения и обслуживается через внешние вводы, наборы базы данных) $f_4$;
- количество внешних интерфейсных файлов – разделяемых с другими программами глобальных файлов (группа логически связанных данных, которая размещена внутри другого приложения и поддерживается им, внутренний логический файл в другом приложении) $f_5$.

Исходные данные для расчета количества функциональных указателей сводятся в таблицу (табл. 1), форма которой приведена ниже (в графах символом $m$ обозначено количество конкретных показателей — при формировании таблицы необходимо данный символ заменить на фактическое число показателей различных характеристик).

Таблица 1

Исходные данные для расчета $FP$ метрик

| Характеристика | Количество с учетом сложности | | | Итого |
|---|---|---|---|---|
| | Низкая | Средняя | Высокая | |
| Внешние вводы | $m$*3 = | $m$*4 = | $m$*5 = | |
| Внешние выводы | $m$*4 = | $m$*5 = | $m$*7 = | |
| Внешние запросы | $m$*3 = | $m$*4 = | $m$*6 = | |
| Внутренние логические файлы | $m$*7 = | $m$*10 = | $m$*15 = | |
| Внешние интерфейсные файлы | $m$*5 = | $m$*7 = | $m$*10 = | |
| Общее количество | | | | |

Значения коэффициентов регулировки сложности $k_i$, зависят от ответов на

следующие вопросы, касающиеся влияния определенных факторов на выполнение функций программного обеспечения:

1. Какое влияние имеет наличие средств передачи данных?
2. Какое влияние имеет распределенная обработка данных?
3. Какое влияние имеет распространенность используемой аппаратной платформы?
4. Какое влияние имеет критичность к требованиям производительности и ограничению времени ответа?
5. Какое влияние имеет частота транзакций?
6. Какое влияние имеет ввод данных в режиме реального времени?
7. Какое влияние имеет эффективность работы конечного пользователя?
8. Какое влияние имеет оперативное обновление локальных файлов в режиме реального времени?
9. Какое влияние имеет скорость обработки данных (вычислений)?
10. Какое влияние имеют количество и категории пользователей?
11. Какое влияние имеет легкость инсталляции?
12. Какое влияние имеет легкость эксплуатации?
13. Какое влияние имеет разнообразие условий применения?
14. Какое влияние имеет простота внесения изменений?

Ответы на каждый вопрос должны быть даны в соответствии со следующими категориями ответов, касающихся влияния конкретных факторов:

- влияние случайное;
- влияние небольшое, эпизодическое;
- влияние среднее;
- влияние важное, значительное,
- влияние основное, существенное.

Значения весовым коэффициентам важности назначают следующим образом:

$k_i = 1$, если на $i$ −й вопрос выбран ответ «влияние случайное»;

$k_i = 2$, если на $i$ −й вопрос выбран ответ «влияние небольшое»;

$k_i = 3$, если на $i$ −й вопрос выбран ответ «влияние среднее»;

$k_i = 4$, если на $i$ −й вопрос выбран ответ «влияние важное»;

$k_i = 5$, если на $i$ −й вопрос выбран ответ «влияние основное».

Тогда метрика количества дефектов может быть определена по формуле

$$DQ = \frac{\text{Количество ошибок}}{FP} = \frac{\text{Количество ошибок}}{F \cdot (0.65 + 0.01 \sum_{i=2}^{14} k_i}$$

После вычисления количества функциональных указателей ***FP*** на его основе формируются косвенные метрики: производительности, качества и другие оценки:

$$\text{Производительность} = \frac{FP}{\text{Затраты}} \ (FP/\text{чел.} - \text{мес.});$$

$$\text{Качество} = \frac{\text{Ошибки}}{FP} \ (\text{Единиц}/FP);$$

$$\text{Удельная Стоимость} = \frac{\text{Стоимость}}{FP} \ (\text{тыс. руб.}/FP);$$

$$\text{Документированность} = \frac{\text{Страниц\_Документа}}{FP} \ (\text{Страниц}/FP).$$

Область применения функциональных указателей - коммерческие информационные системы. Для продуктов с высокой алгоритмической сложностью используется способ оценивания, близкий к оценке на основе функциональных указателей, но несколько модифицированный - метрики свойств ($Features\ Points$). Они применимы к системному и инженерному программному обеспечению, программному обеспечению реального времени и

встроенному программному обеспечению. Для вычисления ***указателя свойств*** добавляется еще одна характеристика - количество алгоритмов. Алгоритм в данном случае определяется как ограниченная программа вычислений, которая включается в общую программу. Для формирования указателя свойств составляется специальная таблица (табл. 2).

Таблица 2.

Исходные данные для расчета указателя свойств

| Характеристика | Количество | Сложность (множитель) | Итого |
|---|---|---|---|
| Вводы | | $m$*4 = | |
| Выводы | | $m$*5 = | |
| Запросы | | $m$*4 = | |
| Логические файлы | | $m$*7 = | |
| Интерфейсные файлы | | $m$*7 = | |
| Количество алгоритмов | | $m$*3 = | |
| | | Общее количество | |

После заполнения таблицы значение каждого указателя свойств вычисляется по формуле, аналогичной для расчета функциональных указателей $FP$. Для сложных систем в реальном масштабе времени это значение оказывается на 25-30 % больше значения, вычисляемого по таблице для количества функциональных указателей.

$FP$ – оценки могут быть пересчитаны в LOC-оценки (табл. 3), поскольку известны примерные соотношения значений указателей с количеством строк кода для различных языков программирования [13].

Таблица 3

Пересчет -оценок в $LOC$-оценки

| Язык программирования | Количество операторов на 1 FP |
|---|---|
| Ассемблер | 320 |
| C | 128 |
| Паскаль | 90 |
| C++ | 64 |
| Java | 53 |
| Visual Basic | 32 |
| Visual C++ | 34 |
| Delphi Pascal | 29 |
| HTML 3 | 15 |
| LISP | 64 |
| Prolog | 64 |

### Связность модулей

Программные средства, разрабатываемые в настоящее время, являются, как правило, большими системами, для которых следует применять меры упрощения еще на этапе проектирования. Один из применяемых методических приемов – разбиение программы на части, реализуемые в виде программных модулей.

*Программный модуль* представляет собой фрагмент описания процесса обработки данных, оформляемый как самостоятельный программный продукт, который может быть применен для использования в описаниях проектируемого процесса. Программные модули физически отделены друг от друга. Кроме того, каждый созданный программный модуль может включаться в состав разных программ, если выполнены условия его использования, заявленные в документации по этому модулю. Это свойство модулей успешно применяется на практике программистами, и именно для этих целей они создают собственные библиотеки ранее разработанных модулей, накапливая знания и сокращая трудозатраты на выполнение будущих проектов.

В связи с этим программные модули могут рассматриваться, с одной стороны, как способ снижения сложности программ, а с другой – как средство борьбы с повторением ранее проделанных действий при программировании. Хороший программный модуль (написанный на более ранних стадиях) проще использовать, модифицируя его, чем создавать заново. В то же время применение модульной структуры не означает, что каждый модуль должен быть абсолютно автономным. Необходимо обеспечить их согласованное исполнение, поэтому программные модули в рамках создаваемого программного средства обязательно должны быть связаны.

Помимо оценки уровня качества методом функциональных указателей, связывающей количество ошибок программы с размером ее программного кода, можно использовать косвенную оценку уровня качества с помощью метрик связности и сцепления модулей в соответствии со следующим положением: *уровень качества программного средства прямо пропорционален силе связности (прочности) и обратно пропорционален силе сцепления модулей.* Для обеспечения необходимого уровня надежности программы разработка качественного программного средства должна предполагать максимизацию связности и минимизацию сцепления модулей.

*Связность* представляет собой меру прочности соединения функциональных и информационных объектов в пределах одного программного модуля. Размещение сильно связанных объектов в одном модуле существенно уменьшает межмодульные взаимосвязи и их взаимное влияние друг на друга.

Для оценки связности используют целочисленную шкалу силы связности в интервале значений от 0 до 10. Каждому из значений силы связности сопоставлен тип связности, который может быть приписан модулю в процессе проектирования. Шкала силы связности использует следующие семь градаций типов связности (рис. 1).

Различные типы связности модулей можно охарактеризовать следующим образом.

1. *Связность по совпадению* (сила связности 0) — справедлива для модулей, в которых отсутствуют явные внутренние связи. Случайно связный модуль похож на логически связный модуль, его объекты не связаны ни потоками данных, ни потоками управления. Тем не менее подзадачи в логически связном модуле относятся к одной категории, а для случайно связанного модуля даже это утверждение является неверным.


***Рис. 1. Шкала связности программных модулей***

2. *Логическая связность* (сила связности 1) — характерна для модулей, содержащих подпрограммы для различных вариантов обращения к модулю, причем каждый раз выполняется одна из подпрограмм модуля. Это означает, что может быть выполнен любой из возможных вариантов действий, но только одна подпрограмма будет запущена, но не все сразу или несколько – последовательно.

3. *Временная связность* (сила связности 3) – характерна для модулей, составные части которых не связаны между собой, но должны быть выполнены в течение определенного периода времени, например инициализации

операционной среды. Например, в конце рабочего дня перед выключением компьютера предусмотрена проверка рабочих файлов на отсутствие вирусов, копирование на устройства хранения и печать отчета о выполненной работе. Последовательность этих действий не имеет особого значения, однако все эти действия предусматриваются к выполнению в конце рабочего дня.

*4. Процедурная связность* (сила связности 5) – типична для модулей, в которых имеет место порядковая связь составляющих их частей, реализующих некоторую предопределенную процедуру обработки данных. При этом типе связности в цепочке обработки данных передается именно управление, т. е. устанавливается последовательность действий, выполняемых программными модулями.

*5. Коммуникационная связность* (сила связности 7) – типична для модулей, когда их составляющие связаны по данным, т. е. используют одну и ту же структуру данных. Типичным примером является использование данных о книгах, содержащихся в информационном объекте ISBN (наименование книги, автор, стоимость), причем разные компоненты этого объекта применяются для решения различных задач одним программным модулем.

*6. Информационная связность* (сила связности 9) - имеет место в случаях, когда выходные данные одной части модуля (подпрограммы) используются как входные данные другой части модуля (подпрограммы). В этом случае объекты охватывают не всю задачу, а только ее часть (подзадачу). Для такого типа связности выходные данные одной из подзадач служат входными данными для следующей (например, цепочка выполнения подзадач печати файла, когда надо сначала открыть файл, затем прочитать его, а потом переслать его на принтер). Особенностью этого типа связности является последовательная передача именно данных, а не управления.

*7. Функциональная связность* (сила связности 10) - модуль как единое целое реализует единственную функцию, поскольку такой программный модуль содержит набор объектов, предназначенных для выполнения одной и только одной задачи (например, для расчета показателей успеваемости у конкретной группы). При вызове такого модуля выполняется полностью только одна задача, причем без выполнения любого другого дополнительного действия.

Специалисты отмечают, что связность типов 1, 2, 3 или 4 свидетельствует о недостатках проектирования архитектуры программы с точки зрения ее тестирования и последующего сопровождения (рис .2).


*Рис. 2. Характеристика типов связанности модулей*

Наличие информационной связности и функциональной связности облегчает тестирование и сопровождение программного обеспечения. Кроме этого, модули, обладающие высокой функциональной связностью, создают хорошие предпосылки для применения прогрессивной технологии повторного использования в процессе будущей разработки программных комплексов.

Выделение типа связности выполняется на основе анализа факторов, характеризующих взаимодействие конкретного программного модуля с остальной частью программного средства, к которому он относится. Для определения типа связности модуля может применяться процедура действий, структура которой приведена на рис. 3.

1. Если модуль реализует единственную прикладную (проблемно-ориентированную) функцию, то тип связности - функциональный. Далее анализ можно прекратить (конец процедуры). В противном случае следует перейти к пункту 2.
2. Если действия внутри модуля связаны, то нужно перейти к пункту 3. Если же действия внутри модуля ничем не связаны, то следует перейти к пункту 5 для дополнительного анализа.

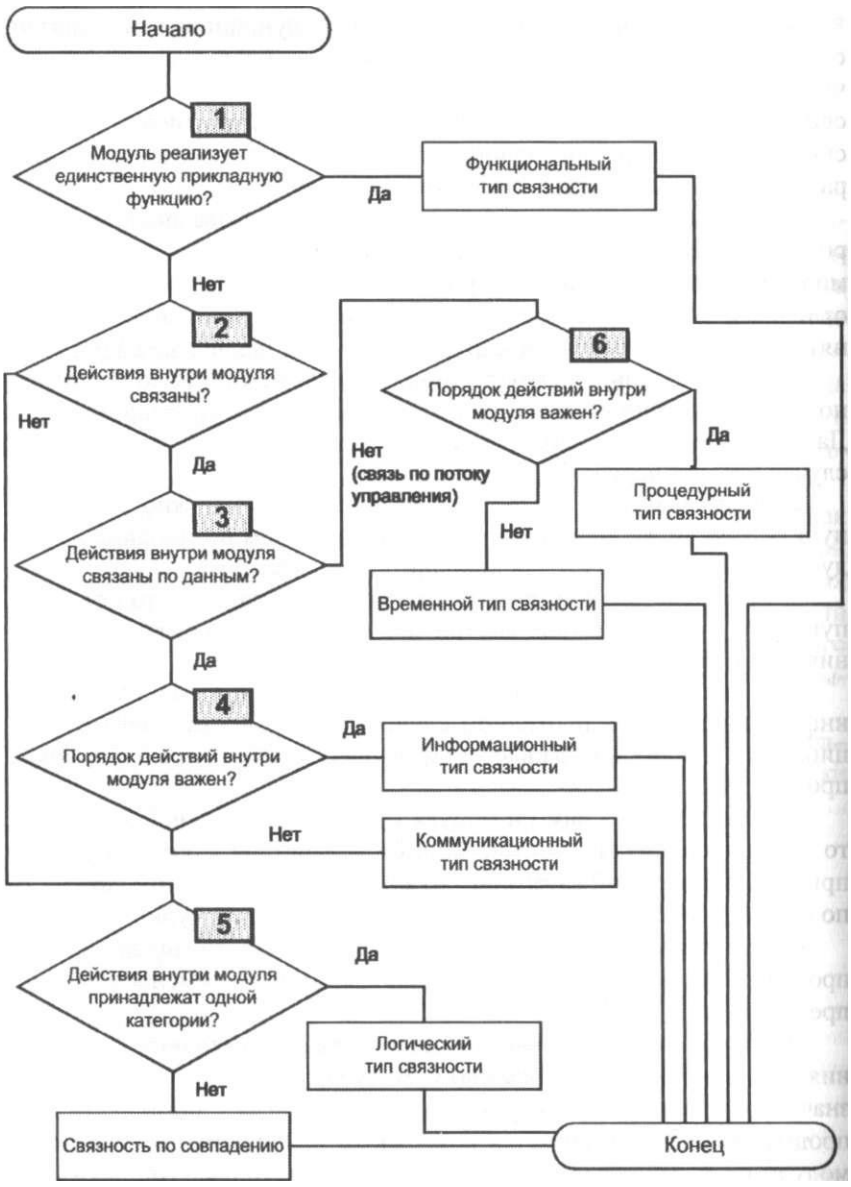


*Рис. 3. Процедура определения типа связности*

3. Если действия внутри модуля связаны по данным, то перейти к пункту 4, а если действия внутри модуля связаны потоком управления - перейти к пункту 6.
4. Если порядок действий внутри модуля важен, то тип связности - информационный, в противном случае тип связности - коммуникационный. Анализ прекращается, и осуществляется переход к концу процедуры.
5. Если действия внутри модуля принадлежат к одной категории, то уровень связности – логический. Если действия внутри модуля не принадлежат к одной категории, то модуль обладает типом связности по совпадению. Далее следует переход к концу процедуры.

6. Если порядок действия внутри модуля важен, то тип связности - процедурный. В противном случае тип связности - временной. Анализ прекращается.

Несмотря на достаточно формализованную методику определения типа связности программного модуля, не всегда удается однозначно определить эту метрику. В случаях, когда с помощью такой процедуры не представляется возможным определить тип связности модуля (т. е. модулю соответствует сразу несколько типов связности), анализируемому программному модулю приписывают тип с наименьшей связностью.

### Сцепление модулей

*Сцепление* определяет меру межмодульной связи, которую для повышения качества программных средств желательно уменьшать. Это один из критериев, который позволяет оценить, насколько хорошо программные модули отделены друг от друга.

Для измерения сцепления используют целочисленную шкалу степени сцепления в интервале значений от 0 до 9. Для каждой степени сцепления определен тип сцепления, который может быть сопоставлен модулю в процессе проектирования. Шкала степени сцепления использует шесть градаций типов связывания (т. е. сцепления) программных модулей (рис. 3.4).

Типы связывания программных модулей можно охарактеризовать следующим образом.


*Рис. 4. Шкала сцепления программных модулей*

1. Сцепление *по данным* (сила сцепления 1). Модуль является вызываемым, и все его входные параметры представлены простыми элементами данных. Это характерно для большинства процедур и функций, имеющих в качестве входных данных формальные параметры. Модули сцеплены по данным, если они взаимодействуют через передачу параметров и при этом каждый параметр является элементарным информационным объектом. В случае небольшого количества передаваемых параметров сцепление по данным обладает наилучшими характеристиками.
2. Сцепление *по образцу* (сила сцепления 3). Модуль является вызываемым, а в качестве параметров используются агрегатные типы (структуры) данных. Данный тип сцепления характерен для программных модулей, вызываемых, например, с передачей сведений о структурированных данных - записей, массивов, элементах баз данных, которые передаются с помощью агрегированных данных. При взаимодействии модулей, сцепленных по образцу, вызывающий модуль передает вызываемому составной информационный объект, т. е. объект, имеющий внутреннюю структуру. Примером составного объекта может быть объект «Клиент», содержащий в себе достаточно много полей (наименование организации, организационно-правовая форма, почтовый адрес, банковские реквизиты и т. д.).
3. Сцепление *по управлению* (сила сцепления 4). Модуль является вызывающим и передает вызываемому модулю список управляющих параметров, явно влияющих на его работу. При этом передается конкретный информационный объект - так называемый *флаг,* который непосредственно определяет особенности последующей работы вызываемого модуля и может существенно влиять на изменение его внутренней логики. В общем случае флаги могут усиливать сцепление модулей и, следовательно, ухудшать качество проекта.
4. Сцепление *по внешним ссылкам* (сила сцепления 5). Модуль имеет

адресные ссылки (указатели) на такой глобальный элемент данных, на который ссылается и другой программный модуль. Такой тип связывания характерен, например, для блоков констант, определенных в качестве глобальных и взаимодействующих со всеми компонентами анализируемого программного средства. Это довольно плохой тип связывания программных модулей. Прежде всего, ошибка в любом модуле, использующем глобальную область, может совершенно неожиданно проявиться в любом другом модуле, который также обращается к этой же глобальной области (глобальные данные не находятся в рамках внутренней структуры модуля и потому не локализованы). Вторая особенность заключается в том, что при таком способе связывания модули обычно используют точные имена (в отличие от формальных параметров). Отсюда следует, что гибкость глобально сцепленных модулей существенно ниже. И наконец, последнее - программные модули, у которых используется много глобальных данных, трудно понимаются при анализе.

5. Сцепление *по общей области* (сила сцепления 7). Модуль разделяет (совместно использует) с другим модулем одну и ту же глобальную структуру данных. Данная ситуация справедлива, например, для нескольких программных модулей, размещенных в границах обобщающего их модуля, причем для всех «внутренних» модулей установлена единая совокупность констант или переменных. Этот тип связывания так же плох для обеспечения качества, как и предыдущий. Причины, снижающие качество программных средств при таком типе связывания, аналогичны недостаткам сцепления по внешним ссылкам.
6. Сцепление *по содержанию* (сила сцепления 9). Модуль напрямую ссылается на содержимое другого модуля (исключая штатную точку входа). При таком типе связывания программный модуль передает управление или выполняет переход в другой модуль, если один модуль ссылается на значения переменных в другом модуле (или

изменяет их) или если один модуль изменяет код другого модуля. Такое сцепление фактически отвергает восприятие программных модулей как локализующих объектов, потому что вынуждает управляющий модуль знать точное содержание и способ реализации другого модуля. Таким образом, этот тип сцепления очень ощутимо может снизить качество проекта. К счастью, только ассемблер позволяет разработчикам применять данный вид сцепления. В большинстве языков программирования такой способ сцепления программных модулей не допускается.

Слабое сцепление между программными модулями служит характеристикой хорошо спроектированной системы по следующим причинам:

- уменьшение связи между двумя модулями способствует уменьшению вероятности появления так называемого волнового эффекта, при проявлении которого возникновение ошибки в одном модуле неминуемо повлияет на работу других модулей;
- низкая степень связи снижает риск появления так называемого эффекта ряби, который проявляется при внесении изменений (например, при исправлении одних ошибок появляются другие, новые ошибки), так как изменение, как правило, влияет на небольшое количество модулей;
- при сопровождении модуля низкая степень связывания может полностью исключить необходимость заботиться о внутреннем содержании других программных модулей, т. е. позволяет обеспечить своеобразную автономность модулей;
- если модули связаны слабо, значительно упрощается контроль их совместного функционирования, а изучение программы (особенно написанной другими разработчиками) существенно упрощает понимание созданной системы.

Достижение слабого сцепления программных модулей происходит не самопроизвольно, для этого нужны направленные действия разработчиков. Слабое сцепление модулей достигается благодаря применению методов, показанных на рис. 5:

- *удаление необязательных связей.* Глубокий анализ структуры и способов связывания модулей, как показывает практика, позволяет избавиться от ряда межмодульных связей, которые на деле оказываются совсем не нужными или без которых вполне можно обойтись;
- *снижение количества необходимых связей.* Искусство программирования к настоящему времени достигло уже достаточно высокого уровня, поэтому профессиональное структурирование разрабатываемой программы и внимательный подход к создаваемому программному средству позволяют достичь требуемого результата;
- *упрощение необходимых связей.* Если уж связывание конкретных модулей необходимо, это не означает, что делать это нужно, не задумываясь о том, каким образом такую связь установить. Даже самые необходимые связи следует хорошо продумывать, чтобы обеспечить наилучшее решение. А наилучшим решением, безусловно, будет самое простое, потому что всем известен принцип «проще — значит надежнее».

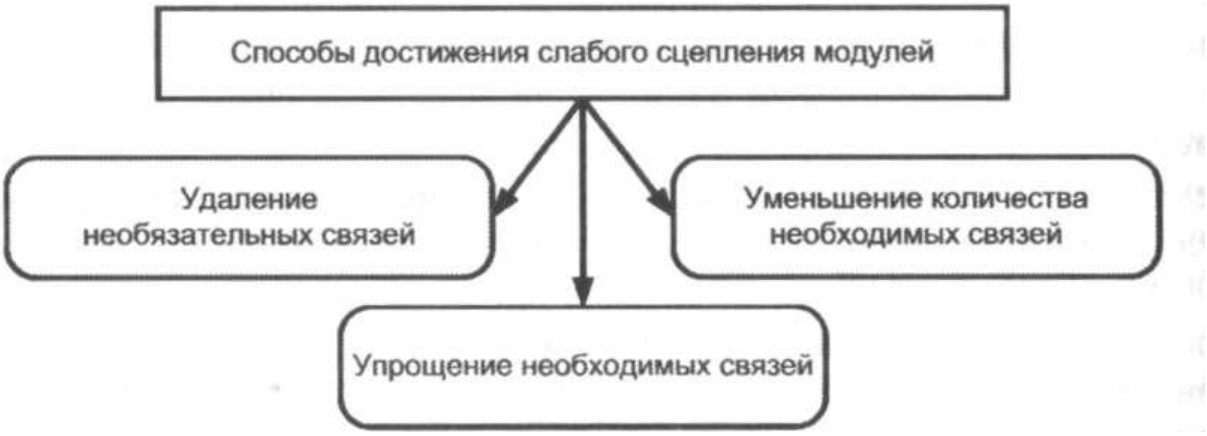


***Рис. 5. Способы снижения степени сцепления модулей***

К практическим рекомендациям по снижению степени сцепления программных модулей можно отнести применение следующих способов (рис. 6).

1. Создание минимальных по количеству параметров межмодульных связей. Снижение количества параметров сопряжения программных модулей позволяет практически пропорционально снизить и степень сцепления.


*Рис. 6. Приемы снижения степени сцепления программных модулей*

2. Создание прямых (а не косвенных) межмодульных связей. При непосредственной передаче параметров взаимосвязь между двумя программными модулями наиболее понятна, и потому становится проще. Учитывая обычную забывчивость программистов даже в отношении собственных разработок, по прошествии некоторого времени человеку гораздо легче вспомнить и понять такие взаимосвязи без дополнительных обращений к другим информационным объектам.
3. Создание локализованных связей. В данном случае имеет значение даже размещение передаваемых параметров. Так, например, значения передаваемых параметров лучше вычислять непосредственно перед вызовом программного модуля. В этом случае проконтролировать их корректность гораздо легче, чем в случае размещения таких параметров на значительном расстоянии от точки их вызова.
4. Создание явных связей. Характерным примером неявной связи является взаимодействие двух программных модулей, построенное на основе модификации области данных, принадлежащих второму модулю. Для того чтобы человек, сопровождающий второй модуль, понял, каким образом модифицируется эта область данных, он будет должен проделать немалую дополнительную работу.
5. Создание гибких связей для облегчения модификаций. Снижение степени сцепления программных модулей или хотя бы гибкость таких связей позволит снизить трудозатраты при любом

совершенствовании созданного программного средства. Жесткое связывание модулей впоследствии потребует больших затрат при переделке, нежели в случае максимальной автономности компонентов программ.

## Примеры

### Задача «Сортировка строк массива»

Задана вещественная матрица размером $N \times M$ элементов. Размер матрицы вводится с клавиатуры ( M не больше 10). Отсортировать строки матрицы по возрастанию их поэлементных сумм. Матрица заполняется случайными числами в диапазоне от -50 до +50.

Пример проведения данной операции приведен ниже.

| Таблица в исходном виде | Сумма | Преобразованная таблица | Сумма |
|---|---|---|---|
| 10 2 3 4 | 19 | 2 3 1 4 | 10 |
| 8 1 1 3 | 13 | 8 1 1 3 | 13 |
| 2 3 1 4 | 10 | 10 2 3 4 | 19 |

Разработать программу и подготовить данные для оценки ее качества на основе применения функционально-ориентированных метрик, для чего определить количество функциональных указателей, уровни связности и сцепления программных модулей.

### Реализация программы

Текст программы для реализации возможного решения поставленной задачи, разработанной с использованием языка программирования C#, приведен на рис. 3.7.

| Номера строк | Строки программы |
|---|---|
| 1 | using System; |
| 2 | class MyMetod |
| 3 | { |
| 4 | public static void sum(double[,] a, int n, int m, double[ ] s) |
| 5 | { |
| 6 | int i, j; |
| 7 | for (i = 0; i < n; i++) |
| 8 | for (s[i] = 0, j = 0; j < m; j++) |
| 9 | s[i] += a[i, j]; |
| 10 | } |
| 11 | public static void change(double[,] a, int m, int row1, int row2) |
| 12 | { |

```
13  double buf;
14  int j;
15  for (j = 0; j < m; j++)
16  {
17  buf = a[row1, j];
18  a[row1, j] = a[row2, j];
19  a[row2, j] = buf;
20  }
21  }
22  public static void sort(double[,] a, int n, int m, double[ ] s)
23  {
24  int i;
25  bool flag;
26  double buf;
27  do
28  {
29  n--;
30  for (flag = false, i = 0; i < n; i++)
31  if (s[i] > s[i + 1])
32  {
33  change(a, m, i, i + 1);
34  buf = s[i];
35  s[i] = s[i + 1];
36  s[i + 1] = buf;
37  flag = true;
38  }
39  } while (flag);
40  }
41  public static void rand(double[,] a, int n, int m, double min,
42  double max)
43  {
44  Random ran = new Random();
45  int i, j;
46  for (i = 0; i < n; i++)
47  for (j = 0; j < m; j++)
48  a[i, j] = min + (max - min) * ran.NextDouble();
49  }
50  public static void print(double[,] a, int n, int m, string fmt)
51  {
52  int i, j;
53  for (i = 0; i < n; i++, Console.WriteLine())
54  for (j = 0; j < m; j++)
55  Console.Write(fmt, a[i, j]);
56  }
57  public static void Main()
```

```
58  {
59  double[,] a;
60  double[ ] s;
61  double min = 1.0, max = 5.0;
62  int n, m;
63  char rep;
64  string sinp;
65  do
66  {
67  Console.Write("Строк: ");
68  sinp = Console.ReadLine();
69  n = int.Parse(sinp);
70  Console.Write("Столбцов: ");
71  sinp = Console.ReadLine();
72  m = int.Parse(sinp);
73  a = new double[n, m];
74  s = new double[n];
75  rand(a, n, m, min, max);
76  print(a, n, m, "{0,8:f2}");
77  Console.WriteLine();
78  sum(a, n, m, s);
79  sort(a, n, m, s);
80  print(a, n, m, "{0,8:f2}");
81  Console.Write("\nДля повтора намите клавишу Y: ");
82  rep = char.Parse(Console.ReadLine());
83  Console.WriteLine();
84  } while (rep == 'Y' || rep == 'y');
85  }
86  }
```

*Рис. 7. Текст программы задачи «Сортировка строк массива»*

## Оценка характеристик программы

Рассмотрим значения $f$ для разработанного решения поставленной задачи, для определения количества функциональных указателей:

$f_1$ — количество внешних вводов (вводов данных пользователем) для рассматриваемой программы составляет 3 (ввод значений переменных $\boldsymbol{n, m, rep}$ – см. рис. 7. строки 68-69, 71-72, 82);

$f_2$ — количество внешних выводов (отчеты, экраны, распечатки, сообщения) составляет 5 (вывод диалоговых сообщений - см. рис. 3.7, строки 67, 70, 81, вывод результатов на экран строки - 76 и 80);

$f_3$ – количество внешних запросов (диалоговых вводов- выводов) равно нулю, так как отсутствуют обращения к внутренним и внешним логическим файлам;

$f_4$ - количество локальных внутренних логических файлов равно нулю, так как в решении такие файлы отсутствуют;

$f_5$ – количество внешних интерфейсных файлов равно нулю, так как в решении такие файлы отсутствуют.

Исходные данные для расчета количества функциональных указателей

сведем в таблицу (табл. 4) с учетом сложности обращений к данным данных. Для внешних вводов, выводов и запросов обращение к 1-4 элементам данных представляет собой низкий уровень сложности, обращение к 5-19 элементам данных - средний уровень сложности, обращение к более 19 элементам - высокий уровень сложности.

Таблица 4

Исходные данные для расчета $FP$ - метрик

| Характеристика | Количество с учетом сложности | | | Итого |
|---|---|---|---|---|
| | Низкая | Средняя | Высокая | |
| Внешние вводы | 3 × 3 = 9 | 0 × 4 = 0 | 0 × 5 = 0 | 9 |
| Внешние выводы | 5 × 4 = 20 | 0 × 5 = 0 | 0 × 7 = 0 | 20 |
| Внешние запросы | 0 × 3 = 0 | 0 × 4 = 0 | 0 × 6 = 0 | 0 |
| Внутренние логические файлы | 0 × 7 = 0 | 0 × 10 = 0 | 0 × 15 = 0 | 0 |
| Внешние интерфейсные файлы | 0 × 5 = 0 | 0 × 7 = 0 | 0 × 10 = 0 | 0 |
| | | | Общее количество | 29 |

Для внутренних и внешних файлов обращение к 1-19 элементам соответствует низкому уровню сложности, к 20-50 соответствует среднему уровню сложности и к более 50 - высокому уровню сложности. В соответствии с уровнем сложности значение характеристики умножается на коэффициент сложности. В табл. 4 коэффициенты сложности расположены справа от знака умножения. При внешних вводах решения программы осуществляется обращение к одному элементу данных, что соответствует низкому уровню сложности (см. рис. 7, строки 69, 72, 82). При внешних выводах также осуществляется обращение к одному элементу данных (см. рис. 7, строка 55). Внешние запросы, внутренние и внешние файлы отсутствуют.

Определим теперь значения коэффициентов регулировки сложности $k_i$, отвечая на поставленные вопросы.

1. Какое влияние имеет наличие средств передачи данных? Ответ: влияние существенное, следовательно, $k_1 = 5$.
2. Какое влияние имеет распределенная обработка данных? Ответ: влияние случайное, отсюда $k_2 = 1$.
3. Какое влияние имеет распространенность используемой аппаратной платформы? Ответ: влияние случайное, тогда $k_3 = 1$.
4. Какое влияние имеет критичность к требованиям

производительности и ограничению времени ответа? Ответ: влияние случайное, поэтому$k_4 = 1$.

5. Какое влияние имеет частота транзакций? Ответ: влияние случайное, следовательно, $k_5 = 1$.
6. Какое влияние имеет ввод данных в режиме реального времени? Ответ: влияние основное, тогда $k_6 = 5$.
7. Какое влияние имеет эффективность работы конечного пользователя? Ответ: влияние среднее, поэтому $k_7 = 3$.
8. Какое влияние имеет оперативное обновление локальных файлов в режиме реального времени? Ответ: влияние случайное, отсюда $k_8 = 1$.
9. Какое влияние имеет скорость обработки данных (вычислений)? Ответ: влияние случайное, поэтому $k_9 = 1$.
10. Какое влияние имеют количество и категории пользователей? Ответ: влияние случайное, тогда $k_{10} = 1$.
11. Какое влияние имеет легкость инсталляции? Ответ: влияние случайное, отсюда $k_{11} = 1$.
12. Какое влияние имеет легкость эксплуатации? Ответ: влияние случайное, следовательно, $k_{12} = 1$.
13. Какое влияние имеет разнообразие условий применения? Ответ: влияние случайное, поэтому $k_{13} = 1$.
14. Какое влияние имеет простота внесения изменений? Ответ: влияние случайное, тогда $k_{14} = 1$

Подсчитываем сумму коэффициентов:

$$\sum_{i=1}^{14} k_i = 5 + 1 + 1 + 1 + 1 + 5 + 3 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 24$$

Подставляем полученные данные:

$$FP = F \cdot (0.65 + 0.01 \cdot \sum_{i=1}^{14} k_i) = 29 \cdot (0.65 + 0.01 \cdot 24) = 25.81$$

Теперь на основе рассчитанного значения количества функциональных указателей $FP$ можно рассчитывать показатели качества, производительности, удельной стоимости и документированности.

Для определения уровня *связности* программных модулей полученного решения воспользуемся алгоритмом процедуры, описанным в теоретической части раздела. Представленное решение включает шесть программных модулей. Рассмотрим каждый модуль в отдельности.

Метод ***sum*** осуществляет суммирование элементов строки массива. Модуль реализует единственную прикладную функцию, следовательно, тип связности - функциональный, сила связности - 10.

Метод ***change*** осуществляет перестановку двух указанных строк массива. Модуль реализует единственную прикладную функцию - тип связности - функциональный, сила связности - 10.

Метод ***sort*** осуществляет сортировку строк массива по возрастанию. Модуль реализует единственную прикладную функцию - тип связности - функциональный, сила связности - 10.

Метод ***rand*** осуществляет заполнение массива случайными вещественными числами заданного диапазона. Модуль реализует единственную прикладную функцию — тип связности — функциональный, сила связности - 10.

Метод ***print*** осуществляет вывод на экран монитора указанный массив. Модуль реализует единственную прикладную функцию - тип связности - функциональный, сила связности - 10.

Метод ***Main*** осуществляет решение поставленной задачи. Модуль реализует не единственную прикладную функцию. Действия внутри модуля связаны. Порядок действия внутри модуля важен - тип связности - коммуникационный, сила связности - 7.

Таким образом, в решении преобладают модули с силой связности 7 и 10, что говорит о достаточно высоком качестве программы, легкой ее тестируемости.

Определим уровень *сцепления* модулей, для чего проанализируем каждый из модулей разработанного решения на предмет межмодульной связи.

Метод ***sum*** является вызываемым, и его входными параметрами являются простые и стуктурные данные (массивы). Таким образом, рассматриваемый модуль имеет сцепление по образцу (сила сцепления 3).

Метод ***change*** является вызываемым, и его входными параметрами

являются простые и стуктурные данные (массивы). Таким образом, рассматриваемый модуль имеет сцепление по образцу (сила сцепления 3).

Метод ***sort*** является вызывающим и одновременнно вызываемым. Поскольку уровень сцепления вызывающего метода выше, то мы не рассматриваем этот метод как вызываемый. Метод ***sort*** осуществляет вызов метода ***change*** с передачей ему списка параметров, влияющих на работу метода. Таким образом, метод ***sort*** имеет сцепление по управлению (сила сцепления 4).

Метод ***rand*** является вызываемым методом и его входными параметрами являются простые и структурные данные (массивы). Таким образом, рассматриваемый модуль имеет сцепление по образцу (сила сцепления 3).

Метод ***print*** является вызываемым методом, и его входными параметрами являются простые и стуктурные данные (массивы). Таким образом, рассматриваемый модуль имеет сцепление по образцу (сила сцепления 3).

Метод ***Main*** является вызывающим и передает вызываемым модулям списки управляющих параметров, явно влияющих на их работу. Таким образом, метод ***Main*** имеет сцепление по управлению (сила сцепления 4).

Сила сцепления программных модулей рассматриваемого решения составляет 3-4, что говорит об уровне качества программы чуть выше среднего.

### Задача «Заполнение массива в шахматном порядке»

Разработать методы:

***print*** – вывод целочисленной матрицы на экран;

***form*** – заполнение одномерного целочисленного массива чередующейся последовательностью двух целых чисел $e0$ и $e1$ значения которых считываются из текстового файла ***in.txt***. Значение первого элемента принять в качестве входного параметра;

***desk*** – заполнение целочисленной матрицы размера $N \times M$ по правилу: строки с четными номерами заполняются, начиная с $e0$, а строки с нечетными номерами - начиная с $e1$ (при квадратной матрице – это образ шахматной доски).

Используя разработанные методы, сформировать и вывести матрицу по указанному правилу. Размер матрицы определить в основной программе.

Разработать программу и подготовить данные для оценки ее качества на основе применения функционально-ориентированных метрик, для чего

определить количество функциональных указателей, уровни связности и сцепления программных модулей.

## Реализация программы

Текст программы для реализации возможного решения поставленной задачи, разработанной с использованием языка программирования C#, приведен на рис. 8.

| Номера строк | Строки программы |
|---|---|
| 1 | using System; |
| 2 | using System.IO; |
| 3 | class MyMetod |
| 4 | { |
| 5 | public static void form(int[ ] a, bool etalon) |
| 6 | { |
| 7 | StreamReader sr=new StreamReader("in.txt"); |
| 8 | int i,e0,e1; |
| 9 | e0 = int.Parse(sr.ReadLine()); |
| 10 | e1 = int.Parse sr.ReadLine()); |
| 11 | for (i = 0; i < a.Length; i++, etalon =!etalon) |
| 12 | a[i] = (etalon) ? e1: e0; |
| 13 | } |
| 14 | public static void desk(int[ ][ ] d) |
| 15 | { |
| 16 | bool etalon = false; |
| 17 | int i; |
| 18 | for (i = 0; i < d.Length; i++, etalon =!etalon) |
| 19 | form(d[i], etalon); |
| 20 | } |
| 21 | public static void print(int[ ][ ] a) |
| 22 | { |
| 23 | int i, j; |
| 24 | for (i = 0; i < a.Length; i++, Console.WriteLine()) |
| 25 | for (j = 0; j < a[i].Length; j++) |
| 26 | Console.Write(" {0,2}", a[i][j]); |
| 27 | } |
| 28 | public static void Main() |
| 29 | { |
| 30 | int[ ][ ] d; |
| 31 | int n,m,i; |
| 32 | char rep; |
| 33 | do |
| 34 | { |
| 35 | Console.Write("Строк: "); |
| 36 | n = int.Parse(Console.ReadLine()); |
| 37 | Console.Write("Столбцов: "); |
| 38 | m = int.Parse(Console.ReadLine()); |
| 39 | d = new int[n][ ]; |
| 40 | for (i = 0; i < d.Length; i++) |
| 41 | d[i] = new int[m]; |
| 42 | desk(d); |
| 43 | print(d); |
| 44 | Console.Write("\nДля повтора намите клавишу Y: "); |
| 45 | rep = char.Parse(Console.ReadLine()); |
| 46 | Console.WriteLine(); |
| 47 | } while (rep = = 'Y' \|\| rep = = 'y'); |
| 48 | } |
| 49 | } |

*Рис. 8. Текст программы заполнения массива в шахматном порядке*

## Оценка характеристик программы

Рассмотрим значения для разработанного решения поставленной задачи:

$f_1$ — количество внешних вводов (вводов данных пользователем) для рассматриваемой программы составляет 3 (ввод значений переменных $n, m, rep$ см. рис. 8, строки 36, 38, 45);

$f_2$ - количество внешних выводов (отчеты, экраны, распечатки, сообщения) составляет 4 (вывод диалоговых сообщений - см. рис. 8, строки 35, 37, 44, вывод результатов на экран – строки 26 и 43);

$f_3$ - количество внешних запросов (диалоговых вводов- выводов) составляет 2 (считывание значений переменных e0и $e1$- см. рис. 8, строки 9 и 10);

$f_4$ - количество локальных внутренних логических файлов 1 (обращение к текстовому файлу $in.txt$ - см. рис. 8, строка 7);

$f_5$ - количество внешних интерфейсных файлов 0, так как в решении такие файлы отсутствуют.

Исходные данные для расчета количества функциональных указателей сведем в таблицу (табл. 5) с учетом сложности обращений к данным данных. Для внешних вводов, выводов и запросов обращение к 1-4 элементам данных представляет собой низкий уровень сложности, обращение к 5-19 элементам данных - средний уровень сложности, обращение к более 19 элементам - высокий уровень сложности.

Таблица 5

Исходные данные для расчета $FP$ – метрик

| Характеристика | Количество с учетом сложности | | | Итого |
|---|---|---|---|---|
| | Низкая | Средняя | Высокая | |
| Внешние вводы | 3 × 3 = 9 | 0 × 4 = 0 | 0 × 5 = 0 | 9 |
| Внешние выводы | 4 × 4 = 16 | 0 × 5 = 0 | 0 × 7 = 0 | 16 |
| Внешние запросы | 2 × 3 = 6 | 0 × 4 = 0 | 0 × 6 = 0 | 6 |
| Внутренние логические файлы | 1 × 7 = 7 | 0 × 10 = 0 | 0 × 15 = 0 | 7 |
| Внешние интерфейсные файлы | 0 × 5 = 0 | 0 × 7 = 0 | 0 × 10 = 0 | 0 |
| | | | Общее количество | 38 |

Для внутренних и внешних файлов обращение к 1-19 элементам соответствует низкому уровню сложности, к 20 - 50 соответствует среднему уровню сложности и к более 50 - высокому уровню сложности. В соответствии с уровнем сложности значение характеристики умножается на коэффициент сложности. В табл. 5 коэффициенты сложности расположены справа от знака

умножения. При внешних вводах решения программы осуществляется обращение к одному элементу данных, что соответствует низкому уровню сложности (см. рис. 3.8, строки 36, 38, 45). При внешних выводах, кроме того, осуществляется обращение к одному элементу данных (см. рис. 3.8, строка 26). При внешних запросах к внутреннему логическому файлу осуществляется обращение к одному элементу (менее 19), количество внутренних файлов один (менее 19) - уровень сложности низкий. Внешние файлы отсутствуют.

Определим теперь значения коэффициентов регулировки сложности $k_i$, отвечая на поставленные вопросы:

1. Какое влияние имеет наличие средств передачи данных? Ответ: влияние существенное, тогда $k_1 = 5$.
2. Какое влияние имеет распределенная обработка данных? Ответ: влияние случайное, поэтому $k_2 = 1$.
3. Какое влияние имеет распространенность используемой аппаратной платформы? Ответ: влияние случайное, следовательно, $k_3 = 1$.
4. Какое влияние имеет критичность к требованиям производительности и ограничению времени ответа? Ответ: влияние случайное, поэтому $k_4 = 1$.
5. Какое влияние имеет частота транзакций? Ответ: влияние случайное, следовательно, $k_5 = 1$.
6. Какое влияние имеет ввод данных в режиме реального времени? Ответ: влияние основное, отсюда $k_6 = 5$.
7. Какое влияние имеет эффективность работы конечного пользователя? Ответ: влияние среднее, поэтому $k_7 = 3$.
8. Какое влияние имеет оперативное обновление локальных файлов в режиме реального времени? Ответ: влияние существенное, тогда $k_8 = 5$
9. Какое влияние имеет скорость обработки данных (вычислений)? Ответ: влияние случайное, следовательно, $k_9 = 1$.
10. Какое влияние имеют количество и категории пользователей? Ответ: влияние случайное, поэтому $k_{10} = 1$.
11. Какое влияние имеет легкость инсталляции? Ответ: влияние

случайное, тогда $k_{11} = 1$.

12. Какое влияние имеет легкость эксплуатации? Ответ: влияние случайное, поэтому $k_{12} = 1$.
13. Какое влияние имеет разнообразие условий применения? Ответ: влияние случайное, отсюда $k_{13} = 1$.
14. Какое влияние имеет простота внесения изменений? Ответ: влияние случайное, следовательно, $k_{14} = 1$.

Подсчитываем сумму коэффициентов:

$$\sum_{i=1}^{14} k_i = 5 + 1 + 1 + 1 + 1 + 5 + 3 + 5 + 1 + 1 + 1 + 1 + 1 + 1 = 28$$

Подставляем полученные данные:

$$FP = F \cdot (0.65 + 0.01 \cdot \sum_{i=1}^{14} k_i) = 38 \cdot (0.65 + 0.01 \cdot 28) = 22.91$$

Теперь на основе рассчитанного значения количества функциональных указателей $\boldsymbol{FP}$ можно рассчитывать показатели качества, производительности, удельной стоимости и документированности.

Для определения уровня *связности* программных модулей полученного решения воспользуемся алгоритмом процедуры, описанным в теоретической части раздела. Представленное решение включает четыре программных модуля. Рассмотрим каждый модуль в отдельности.

Метод $\boldsymbol{form}$ осуществляет заполнение одномерного массива чередующимися значениями переменных $e\backslash$ и $eo$. Модуль реализует единственную прикладную функцию - тип связности - функциональный, сила связности 10.

Метод $\boldsymbol{desk}$ осуществляет заполнение двумерного массива двумя чередующимися целыми значениями в шахматном порядке. Модуль реализует единственную прикладную функцию - тип связности - функциональный, сила связности - 10.

Метод $\boldsymbol{print}$ осуществляет вывод строк массива на экран монитора. Модуль реализует единственную прикладную функцию - тип связности - функциональный, сила связности - 10.

Метод $\boldsymbol{Main}$ осуществляет решение поставленной задачи. Модуль

реализует не единственную прикладную функцию. Действия внутри модуля связаны. Порядок действия внутри модуля важен - тип связности - коммуникационный, сила связности — 7.

Таким образом, в решении преобладают модули с силой связности 7 и 10, что говорит о достаточно высоком качестве программы и ее легкой тестируемости.

Определим уровни *сцепления* модулей, для чего проанализируем каждый из модулей разработанного решения на предмет межмодульной связи.

Метод ***form*** является вызываемым, и его входными параметрами являются простые и структурные данные (массивы). Таким образом, рассматриваемый модуль имеет сцепление по образцу (сила сцепления 3).

Метод ***desk*** является вызывающим и одновременнно вызываемым. Поскольку уровень сцепления вызывающего метода выше, то мы не рассматриваем этот метод как вызываемый. Метод ***desk*** осуществляет вызов метода ***form*** с передачей ему списка параметров, влияющих на работу метода. Таким образом, метод ***desk*** имеет сцепление по управлению (сила сцепления 4).

Метод ***print*** является вызываемым методом, и его входными параметрами являются структурные данные (массивы). Поэтому рассматриваемый модуль имеет сцепление по образцу (сила сцепления 3).

Метод ***Main*** является вызывающим и передает вызываемым модулям списки управляющих параметров, явно влияющих на их работу. Следовательно, метод ***Main*** имеет сцепление по управлению (сила сцепления 4).

Сила сцепления программмых модулей рассматриваемого решения составляет 3-4, что говорит об уровне качества программы чуть выше среднего.

## Задачи для самостоятельного решения

В задачах, предлагаемых для самостоятельного решения, необходимо с целью оценки качества программы выполнить следующее:

- разработать программу, реализующую предложенные в условии действия;
- провести расчет количества функциональных указателей;
- оценить уровень связности и силу сцепления программных модулей;

- провести анализ полученных результатов, сформировав содержательные выводы.

Задача 1. Функция должна переворачивать строку, передаваемую ей в качестве параметра, в зеркальное положение.

Задача 2. Дано натуральное число n. Вывести на экран число, которое получится после записи цифр числа n в обратном порядке. Для получения нового числа составить функцию.

Задача 3 . Функция должна подсчитать количество минимальных элементов в целочисленной матрице.

Задача 4. Функция должна преобразовывать строку восьмеричных цифр в эквивалентное ей целое десятичное число.

Задача 5 . Функция $\boldsymbol{minmax}(\boldsymbol{x}, \mathrm{y})$ должна присваивать первому параметру большее, а второму - меньшее из значений $x$ и y. Используя эту функцию, перераспределить введенные значения переменных $\mathrm{A}, \mathrm{B}, \mathrm{C}$ таким образом, чтобы стало $\mathrm{A} < \mathrm{B} < \mathrm{C}$.

Задача 6. Даны две квадратные матрицы. Напечатать ту из них, которая имеет минимальный «след», т. е. сумму элементов главной диагонали. Разработать функцию для нахождения следа матрицы и функцию вывода матрицы на экран.

Задача 7. Функция $\boldsymbol{compress}()$ должна сжимать строку, удаляя из нее все пробелы.

Задача 8. Функция должна подсчитывать, сколько раз в заданной строке встретился указанный символ.

Задача 9. Функция должна находить в массиве минимальный по модулю элемент и заменять им все элементы с нечетными номерами.

Задача 10. Функция должна в прямоугольной матрице находить сумму элементов -й строки.

Задача 11. Функция должна по заданному натуральному числу определять количество цифр в нем и их сумму.

Задача 12. Функция должна по заданной строке $\boldsymbol{Str}$, содержащей буквы и цифры, формировать новую строку, состоящую только из цифр, входящих в $\boldsymbol{Str}$.

Задача 13. Функция должна подсчитывать количество положительных

элементов в массиве.

Задача 14. Определить, является ли данная матрица ортонормированной, т. е. такой, в которой скалярное произведение каждой пары различных строк равно нулю, а скалярное произведение каждой строки на себя равно единице.

Задача 15. Функция должна заменять знаки двоеточия на тире в заданной строке, начиная с указанной позиции.

Задача 16. Выяснить, сколько простых чисел находится в интервале $[\boldsymbol{n}, \boldsymbol{m}]$, и вывести их на экран. Для определения, является ли очередное число простым, составить функцию.

Задача 17. Функция должна вычислять площадь треугольника, заданного координатами вершин.

Задача 18. Функция должна находить в прямоугольной матрице номер строки, имеющей максимальную сумму элементов.

Задача 19. Функция должна преобразовывать строку двоичных цифр в эквивалентное ей целое десятичное число.

Задача 20. Функция должна в строке, передаваемой ей в качестве параметра, заменять каждый второй элемент на заданный символ.

Задача 21. Функция должна преобразовывать строку шестиричных цифр в эквивалентное ей целое десятичное число.

Задача 22. Функция, которая должна преобразовывать все прописные символы в строчные.

Задача 23. Функция должна в матрице находить сумму элементов -го столбца.

Задача 24. Функция должна преобразовывать строку десятичных цифр в эквивалентное ей двоичное число.

Задача 25. Программа должна, в качестве входных данных, принимать пятизначные числа и определять, является ли введенное пятизначное число палиндромом.