# Содержание

Введение

1. Теоретическая часть

1.1 Модульное программирование

1.1.1 Процедуры и функции

1.1.2 Модули

1.1.3 Открытые массивы и строки

1.1.4 Нетипизированные параметры

1.2 Объектно-ориентированное программирование

Заключение

Список литературы

# Введение

На заре развития программирования (в историческом смысле это было совсем недавно) компьютерная программа рассматривалась как результат научного творчества и искусства. Программистов было совсем немного, а их труд считался сродни волшебству. В настоящее время ситуация изменилась: к программе подходят к результату сложного технологического процесса, а программисты – это прежде всего настоящие профессионалы своего дела, искусные, т.е. квалифицированные, мастера.

В курсовой работе поставлена задача рассмотреть в теоретической части разработку программных средств для решения экономических задач, а именно рассмотреть модульное и объектно-ориентированное программирование, рассмотреть функции и назначения каждого из типа программирования. В практической части поставлены следующие задачи:

1. Построить таблицы по приведенным данным.

2. Выполнить расчет отклонения фактических показателей от плановых в абсолютной и относительной форме, подвести итоги.

3. Результаты представить в виде консолидированной таблицы.

4. Сформировать и заполнить форму сводной ведомости.

5. Результаты представить в графическом виде.

Объектом изучения является модульное и объектно-ориентированное программирование.

# 1. Теоретическая часть

## 1.1 Модульное программирование

Большие программы обычно разрабатывают и отлаживают по частям. Целесообразно при этом, каждая такая часть, называемая подпрограммой, была оформлена так, чтобы ее можно было использовать при решении аналогичной подзадачи в той же программе или даже при решении других задач. В Borland Pascal реализованы два типа подпрограмм: процедуры и функции.

### 

### 1.1.1 Процедуры и функции

Процедуры и функции представляют собой относительно самостоятельные фрагменты программ, соответствующим образом оформленные и снабженные именем (программные блоки). По правилам Borland Pascal программные блоки – такие же ресурсы, как типы и переменные. Соответственно, они также должны быть описаны перед использованием в разделе описаний программного блока, который их использует (основной программы или вызывающей подпрограммы). Каждый блок имеет такую же структуру, как основная программа, т.е. включает заголовок, раздел описаний и раздел операторов, но заканчивается не точкой, а точкой с запятой.

Заголовок блока определяет форму вызова программы. В разделе описаний блока объявляют внутренние локальные ресурсы блока (переменные, типы, внутренние подпрограммы). Раздел операторов содержит инструкции подпрограммы в операторных скобках begin … end.

Заголовки процедур и функций описываются по-разному. В отличие от процедуры функция всегда возвращает в точку вызова скалярное значение, адрес или строку. Тип возвращаемого результата описывается в заголовке функции.

Данные для разработки процедуры и функции получают из вызвавшей их основной программы. Для размещения рабочих полей подпрограммы могут объявлять новые типы и переменные в собственном разделе описаний. Результаты же они обычно должны возвращать вызвавшей программе или подпрограмме.

Из основной программы данные могут быть получены:

* Неявно – с использованием глобальных констант и переменных;
* Явно – через параметры.

**Неявная передача данных в подпрограммы.** Каждой подпрограмме доступны все ресурсы программного блока, в разделе описаний которого эта подпрограмма объявлена. Ресурсы же основной программы доступны в любой подпрограмме. Они получили название *глобальных.*

В свою очередь *локальные* ресурсы, объявлены в разделе описания подпрограммы, из программного блока, в разделе описания которого она определена, не доступны. В том случае, если в подпрограмме объявляется ресурс, имя которого совпадает с именем глобального ресурса, соответствующий глобальный ресурс в подпрограмме становится не доступным, «перекрывается».

**Передача данных через параметры.** Список параметров описывается в заголовке подпрограммы. Параметры, перечисленные в этом списке, получили название *формальных,* так как для их размещения не отводится память. При обращении к подпрограмме для каждого параметра должно быть указано *фактическое* значение – литерал, константа или переменная того же типа, что и формальный параметр. Несоответствие типов и количества формальных и фактических параметров выявляется компилятором (или компоновщиком, если вызов подпрограммы происходит из другого модуля). Нарушение порядка следования фактических параметров, если это нарушение не связано с несовпадением количества параметров или их типов, приводит к нарушению логики работы программы и часто может быть обнаружено только при тестировании программы.

В Borland Pascal параметры в подпрограмму могут передаваться тремя способами:

* Как значения – в подпрограмму передаются копии значений параметров, и никакие изменения этих копий не возвращаются в вызывающую программу;
* Как переменные – в программу передаются адреса фактических параметров, соответственно все изменения этих параметров в подпрограмме на самом деле происходят с переменными, переданными в качестве фактических параметров; такие параметры при описании помечаются служебным словом var; в качестве фактических значений параметров – переменных нельзя использовать литералы;
* Как изменяемые переменные (именованные константы) – в подпрограмму, так же как и в предыдущем случае, передаются адреса фактических параметров, но при попытке изменить значение параметра компилятор выдает сообщение об ошибке; такие параметры при описании помечаются служебным словом const.

**Вызов процедур и функций.** И процедура, и функция, используя параметры – переменные, могут изменять значения переменных основной программы. Но как отмечалось выше, функция отличается от процедуры тем, что кроме изменения значений параметров-переменных всегда возвращает в точку вызова скалярное значение, строку или указатель. Поэтому в теле функции обязательно наличие специальной переменной с именем функции, которой должно присваиваться значение. Именно это значение и будет возвращено в место вызова функции в качестве его результата. Вызов функции, таким образом, можно осуществлять в составе выражений везде, где возможно использование выражений ( в операторе присваивания, в операторе вывода и т.д.), например:

**<**переменная**>**:=**<**имя функции**>**(**<**фактические параметры**>**).

Процедура же должна вызываться отдельным оператором, состоящим из имени процедуры и списка фактических параметров.

Вызов процедуры и функции по-разному изображается на схеме алгоритма: вызов функции – в блоке «процесс» или блоке вывода, а для вызова процедуры используется специальный блок «предопределенный процесс». Схемы алгоритмов же самих подпрограмм в обоих случаях оформляются отдельно, причем вместо слова «начало» указывают имя подпрограммы, а вместо слова «конец» - указывают слово «возврат» или «return».

### 1.1.2 Модули

При разработке больших программ целесообразно часть подпрограмм и других ресурсов, таких, как переменные, константы, описания типов, собирать вместе и компилировать отдельно от основной программы в виде библиотек ресурсов или модулей.

***Модуль*** *–* это автономно компилируемая коллекция программных ресурсов, предназначенная для использования другими модулями и программами.

Все ресурсы модуля делятся на две группы: *внешние –* предназначенные для использования другими программными единицами, и *внутренние –* рабочие ресурсы данного модуля.

Структура модуля выглядит следующим образом:

***Unit <***имя модуля***>***

***Interface***

<интерфейсная секция>

***[Begin***

<секция инициализации>]

***End.***

Имя модуля должно совпадать с именем файла, в котором он содержится. Результат компиляции модуля помещается в файл с тем же именем и расширением .tpu.

***Примечание.***Среда языка Borland Pascal предусматривает три режима компиляции программы, использующей модули:

* **Compile** – компилируется только основная программа, все модули должны быть предварительно откомпилированы в файлы <имя модуля> .tpu и размещены либо в текущем каталоге, либо в одном из каталогов, указанных как источники файлов .tpu в настройках среды;
* **Make** – модули, для которых не обнаружены файлы .tpu, компилируются из соответствующих файлов .pas, которые должны находиться в текущем каталоге или в каталогах, указанных в настройках среды в качестве источников исходных файлов модулей;
* **Build** – все ранее откомпилированные модули .tpu игнорируются и все модули компилируются из своих исходных файлов заново.

В процессе отладки модулей целесообразно использовать режим Build, а при отладке программы – режим Compile

Интерфейсная секция содержит объявление ресурсов (в том числе заголовки подпрограмм), к которым возможны обращения извне.

Секция реализации содержит описание подпрограмм, объявленных в интерфейсной секции, и описание внутренних ресурсов модуля (локальных переменных, типов, подпрограмм). Обращение к этим ресурсам возможно только из подпрограмм, описанных в том же модуле.

Секция инициализации содержит операторы, которые выполняют следующие действия, необходимые для нормальной работы процедур модуля (например, открываются файлы, инициализируют некоторые переменные и т.п.). Операторы секции инициализации выполняются один раз (при подключении модуля) до начала выполнения основной программы. Эта секция в модуле может отсутствовать.

Программа, которая использует ресурсы нескольких модулей, должна в области описаний содержать спецификацию используемых модулей:

Uses <имя модуля 1>, <имя модуля 2>,

В спецификации Uses необходимо указывать только те модули, ресурсы которых данная программная единица (программа или модуль) использует непосредственно. Если подключаемый модуль использует другие модули, то их подключение уже описано в нем. Секции инициализации подсоединяемых модулей выполняются в порядке их подключения.

В виде модулей в Borland Pascal реализованы библиотеки подпрограмм, использование которых существенно упрощает разработку программ.

Вместе с системой программирования на Borland Pascal поставляются следующие библиотеки:

* **System** – основная библиотека – содержит описание всех стандартных процедур и функций, таких, как математические функции, функции преобразований, процедуры и функции обработки строки и т.п. Ресурсы данной библиотеки доступны любой программе без специального указания.
* **Crt** – библиотека управления экраном в текстовом режиме – содержит описание переменных, констант и процедур и функций, обеспечивающих управление экраном, клавиатурой и динамиком.
* **Graph** – библиотека управления экраном в графическом режиме – содержит описание переменных, констант и процедур и функций, обеспечивающих управление экраном в графическом режиме.
* **Dos** – библиотека организации взаимодействия с операционной системой MS **DOS** – содержит описание процедур и функций, обеспечивающих обращение к функциям операционной системы.

Поставляемые вместе с описанными модули Turbo3, Printer, Graph3, Overlay устарели и практически не используются.

При разработке собственных библиотек программисты стремятся создавать подпрограммы, имеющие широкую область применения, для чего используют специальные средства объявления параметров: открытые массивы и строки, нетипизированные параметры и параметры процедурного типа.

### 1.1.3 Открытые массивы и строки

По правилам Borland Pascal размерность любого массива, передаваемого в качестве параметра, должна быть определена. Следовательно, без использования специальных средств применимость процедур и функций, имеющих параметры-массивы, существенно ограничивается. Чтобы снять ограничение размерности для параметров – одновременных массивов, можно использовать открытые массивы.

***Открытый массив*** – это конструкция описания типа массива без указания типа индексов.

Такое определение возможно только при описании формальных параметров подпрограммы. Применяя открытые массивы, следует помнить, что индексы параметров, описанных как открытые массивы, всегда начинаются с нуля. Реальное количество элементов фактического параметра массива можно определить двумя способами. Во-первых – передать через дополнительные параметры, во-вторых – использовать специальные функции.

**High (<**идентификатор массива**>)** – для обычного массива возвращает верхнюю границу индекса массива, для открытого – максимальное значение индекса.

**Low (<**идентификатор массива**>)** – для обычного массива возвращает нижнюю границу индекса массива, для открытого – ноль.

### 1.1.4 Нетипизированные параметры

ВBorland Pascal допускается использовать параметры, тип которых не указан. Такие параметры могут передаваться в подпрограмму только по ссылке, так как в этом случае в подпрограмму реально передается адрес параметра.

Безусловно, для того чтобы подпрограмма могла выполнять какие-либо действия с этим параметром, она должна как-то назначить ему тип.

Для приведения нетипизированного параметра к определенному типу можно использовать:

* Автоопределенное преобразование типов;
* Наложенное описание переменной определенного типа.

При автоопределенном преобразовании типов тип выражения указывают явно, например:

Procedure Proc (Var: а);

b: = Integer (a) + 10;

Для наложения переменной определенного типа используют описание с absolute, например:

Procedure Proc (Var: а);

Var r : real absolute a;

## *При этом переменная r оказывается в памяти размещенной в том же месте, что и нетипизированный параметр а, и, соответственно, любое изменение r приведет к изменению а.*

## 1.**2. Объективно-ориентированное программирование**

Современная технология разработки программных продуктов, в том числе и операционной системы Windows, базируется на концепции объектно-ориентированного программирования, в которой выдерживается единый подход к данным и программам. В основе всего лежит понятие объекта, который объединяет в себе как алгоритмы, так и данные, обрабатываемые этими алгоритмами. В результате упрощается не только разработка программ, но и технология работы пользователя, которому представляется возможность при работе в интерактивном (диалоговом) режиме применять наглядные графические инструменты и различные подсказки.

Объектно-ориентированное программирование стало необычайно популярным в последние годы. Оно определяет новое понимание процесса вычисления, а также то, как можно структурировать информацию внутри компьютера.

Формализованной основой ООП является модель с рядом базовых категорий:

* **Абстрагирование** – процесс выделения наиболее существенных характеристик некоторого объекта, которые отличают его особенности при дальнейшем анализе и использовании. При этом абстрагирование концентрирует внимание на внешних особенностях объекта, позволяя в значительной степени игнорировать детали реализации этих особенностей. Именно выбор эффективного набора абстракций для конкретной предметной области и представляет собой главную задачу ООП.
* **Инкапсуляция** – процесс разделения всех, составляющих объект элементов, на две категории, определяющие его внутреннюю организацию и внешнее поведение. Инкапсуляция позволяет изолировать внешний интерфейс объекта (доступный для взаимодействия с другими объектами) от внутренней реализации объекта. При этом инкапсуляция позволяет связать структуры данных с функциями (методами), предназначенными для манипулирования этими данными, и скрыть их от внешней среды.
* **Модульность** – свойство системы, определяющее возможность ее декомпозиции на ряд слабо связанных между собой модулей. Совместно с инкапсуляцией модульность создает барьеры между абстракциями.
* **Иерархия** – расположение (упорядочивание) абстракций по ряду уровней. В рамках сложных систем основными видами иерархических структур являются
* структура классов (иерархия по номенклатуре введенных классов) и структура объектов (иерархия по составу).

Абстрагирование и инкапсуляция являются взаимодополняющими операциями: абстрагирование акцентирует влияние на внешних особенностях объекта, а инкапсуляция (или, иначе, ограничение доступа) не позволяют объектам-пользователям различать внутреннее устройство объекта. Такой подход позволяет программисту не знать частных деталей реализации программной системы и программировать, используя объекты с закрытой внутренней организацией. Инкапсуляция предполагает введение нового механизма структурирования и оптимизации данных – класса.

Помимо описанных основных обязательных категорий объектной модели, в ряде случаев, важное значение имеют три дополнительных категорий:

* **Типизация** – ограничение, накладываемое на класс объектов и препятствующее взаимозаменяемости различных классов (использованию объектов одного класса вместо другого);
* **Параллелизм** – свойство объектов находиться в активном или пассивном состоянии и, соответственно, возможность отличать друг от друга активные и пассивные объекты;
* **Устойчивость** – свойство объектов существовать во времени (вне зависимости от породившего данный объект процесса) и/или в пространстве (при перемещении объекта из области, в которой он был создан, в другую область).

К концептуальным понятиям в ООП относятся объекты и классы.

**Объект** определяется как осязаемая реальность (предмет или явление), характеризующаяся определенным поведением. Объект описывается состоянием, поведением и индивидуальностью. Объекты со схожей структурой и поведением входят в общий для них класс. Состояние объекта определяется перечнем всех возможных свойств этого объекта и текущими значениями каждого из этих свойств. Поведение объекта характеризуется его воздействием на другие объекты и изменением собственного состояния в результате воздействий со стороны других объектов. В ООП операции, выполняемые над объектом, называют методами; они являются составной частью определения класса. В общем случае объектом может быть и константа, и переменная, и процедура, и процесс.

**Класс** определяется как множество объектов, обладающих внутренними свойствами, присущими любому объекта класса и проявляющимися в общности структуры и поведения. При этом спецификация (определение) класса проводится путем определения его внутренних свойств, играющих роль классообразующих признаков. Определение классов и объектов является одной из наиболее сложных задач в объектно-ориентированном проектировании.

Важное значение для классов имеют свойства наследования и полиморфизма.

**Наследование** означает создание новых, производных классов посредством использования данных и функций от одного или нескольких ранее определенных классов. Наследование позволяет разрабатывать сложные классы, продвигаясь от общего к частному.

**Полиморфизм** (многоформенность) означает возможность принадлежности класса одновременно более чем одному типу. Однако при этом предполагается, что в рамках различных типов иерархической структуры класс имеет возможность несколько различным образом проявлять себя.

Объектно-ориентированная система в наибольшей мере отражает возможность ее последующей эволюции. Наследование и полиморфизм позволяют создавать новые производные классы на основе базовых классов. При этом потомки наследуют характеристики родительских классов и добавляют необходимые собственные структуры данных и методы. Подобный подход, при котором заново описываются только различия и уточнения, позволяет в значительной степени сократить время и усилия, требующиеся на разработку новых программ.

Объектно-ориентированный подход, как и другие концептуальные решения в процессе развития технологии программирования, явился ответом на поиск путей снижения стоимости разработки ПО. Практика программирования последних лет показывает, что удачное применение механизмов ООП позволяет построить множество объектов различных классов, которые многократно могут быть использованы без какой-либо доработки при выполнении многих проектов различной сложности, что снижает затраты на их реализацию.

В целом объектно-ориентированный подход к разработке программ интегрирует в себе как методы структуризации управления, так и структуризации данных.

Объектно-ориентированные языки программирования характеризуются наличием механизма, реализующего отношение класс-подкласс, связанного с использованием возможности наследования свойств.

В качестве преимуществ ОПП отмечают:

* Повышение уровня унификации разработки и возможность повторного использования не только программ, но и проектов, что ведет к созданию среды разработки и переходу к сборочному созданию ПО (переход от непосредственного программирования к проектированию);
* Естественность объектной модели, ориентированной на человеческое восприятие мира, а не на компьютерную реализацию;
* Уменьшение риска создания чрезмерно сложных систем ПО, поскольку изначально подход предполагает эволюционный путь развития системы во времени на базе исходных, относительно несложных подсистем.

К основным недостаткам ООП относят значительные начальные затраты и некоторое снижение производительности функционирования ПО. Эффект от применения ООП начинает сказываться после разработки нескольких проектов и накопления повторно используемых компонентов. При этом следует иметь в виду и значительные психологические трудности, связанные с переходом к ООП.

Фактически, основное различие между объектно-ориентированным и «обычным» программированием состоит в том, что при обычном программировании программисты ограничены встроенными абстракциями, в то время как при использовании объектно-ориентированного программирования они могут определять свои собственные абстракции. Объектно-ориентированное проектирование – дело чрезвычайно сложное. Необходим большой опыт и здравый смысл, чтобы решить, что же заслуживает того, чтобы стать объектом. В целом ООП использует в качестве основной конструкции не только абстрактный тип данных, оно является более общим в целом и простирается до абстрагирования внешних устройств, моделей реального мира и т.д.

Следует отметить, что интегрированный пакет программ MS Office также построен на основе объектно-ориентированного программирования, и его можно рассматривать как семейство классов и объектов.

# 

# Заключение

Большие программы обычно разрабатывают и отлаживают по частям. Целесообразно при этом, каждая такая часть, называемая подпрограммой, была оформлена так, чтобы ее можно было использовать при решении аналогичной подзадачи в той же программе или даже при решении других задач. В Borland Pascal реализованы два типа подпрограмм: процедуры и функции.

***Модуль*** *–* это автономно компилируемая коллекция программных ресурсов, предназначенная для использования другими модулями и программами.

Все ресурсы модуля делятся на две группы: *внешние –* предназначенные для использования другими программными единицами, и *внутренние –* рабочие ресурсы данного модуля.

Современная технология разработки программных продуктов, в том числе и операционной системы Windows, базируется на концепции объектно-ориентированного программирования, в которой выдерживается единый подход к данным и программам. В основе всего лежит понятие объекта, который объединяет в себе как алгоритмы, так и данные, обрабатываемые этими алгоритмами. В результате упрощается не только разработка программ, но и технология работы пользователя, которому представляется возможность при работе в интерактивном (диалоговом) режиме применять наглядные графические инструменты и различные подсказки.

Объектно-ориентированное программирование стало необычайно популярным в последние годы. Оно определяет новое понимание процесса вычисления, а также то, как можно структурировать информацию внутри компьютера.

**Список литературы**

1. Бадд Тимоти Объектно-ориентированное программирование в действии/ [Пер. с англ. А Берднокова; Гл. ред. В. Усманов]. – СПб: Питер, 1997. – 460 с.: ил.
2. Басс Лен Архитектура программного обеспечения на практике [пер. с англ.] / Л. Басс 2-е – изд. – СПб: Питер, 2006. – 574 с.
3. Боровцов Е.Г. Введение в объектно-ориентированное программирование.: Учеб. пособие по курсу «Технология программирования для студентов специальностей 2204» - «Программное обеспечение вычислительной техники»/ Гос. ком. Рос. Федерации по высшему образованию, Алт. гос. техн. ун-т им. И.И. Ползунова. – Барнаул: Изд-во АлтГТУ, 1996. – 79 с.
4. Бутаков С.В. Высокоуровневые методы информатики и программирования. Учеб. пособие. : Министерство образования и науки Российской Федерации, Федеральное агентство по образованию, Алт. академия экономики и права – Барнаул: изд-во ААЭП, 2005. – 72с.
5. Иванова Г.С. Основы программирования [Учеб. для вузов] 2-е – изд. перераб. и дополнено. – М.: изд – во МГТУ им. Н.Э. Баумана, 2002. – 415 с. ил.
6. Непейвода Н.Н. Стили и методы программирования курс лекций: учеб. пособие: М.: 2005. – 316 с.: ил.
7. Окулов С.М. Основы программирования. М.: Лаборатория базовых значений 2002. – 424 с.
8. Пол Айра Объектно-ориентированное программирование на С++/ Пер. с англ. Д. Ковальчука. – 2-е – изд. – М; СПб: БИНОМ; Невский диалект 1999. – 461 с.
9. Солодкий О.Г. Основы программирования: сборник задач; М-во образования и науки Рос. Федерации по образованию, Алт. акад. экономики и права. – Барнаул: Изд. ААЭП, 2005. – 83 с.
10. Хореев П.Б. Технологии объектно-ориентированного программирования; учеб. пособие; - М.: Academia, 2004. – 447 с.