ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

Методические рекомендации для самостоятельной работы обучающихся по дисциплине

Б1.В.ДВ.13.02 Прикладные задачи программирования

Направление подготовки 35.03.06 Агроинженерия

Профиль образовательной программы «Электрооборудование и электротехнологии»

Форма обучения: очная

СОДЕРЖАНИЕ

1. ОРГАНИЗАЦИЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Организационно-методические данные дисциплины

№ п.п.	Наименование темы	Общий объем часов по видам самостоятельной работы				
		подготовка курсового проекта (работы)	подготовка реферата/эссе	индивидуальные домашние задания (ИДЗ)	самостоятельное изучение вопросов (СИВ)	подготовка к занятиям (ПкЗ)
1	2	3	4	5	6	7
1	Цели использования компьютеров при решении прикладных задач. Задачи и особенности прикладного программирования				1	
2	Основные инструменты прикладного программиста. Язык программирования - главный инструмент прикладного программиста. Выбор языка программирования				1	
3	Технологии прикладного программирования: цели, задачи и основные принципы и инструменты. Алгоритмическая и объектно-ориентированная декомпозиция. Принципы объектно-ориентированного анализа: абстрагирование, инкапсуляция, наследование, полиморфизм, модульность, сохраняемость, параллелизм. Объекты и типы объектов. Атрибуты и типы атрибутов.				3	
4	Экземпляры и состояния. Жизненный цикл и поведение объектов: сообщения, события, методы, действия				1	
5	Структура программы на языке C++. Проект. Компиляция программы и сборка исполняемого модуля. Размещение программы и данных в памяти. Структура исполняемого модуля. Переменные: объявление, определение, инициализация. Переменные: значение, указатель, ссылка. Время жизни, области видимости и классы памяти переменных				2	
6	Динамическое размещение данных в памяти. Составные типы данных. Массивы - как пример гомогенной структуры данных: размещение в памяти, доступ к элементам. Одномерные и многомерные массивы. Структуры - как пример гетерогенной структуры данных. Реализация вычислительных операций. Арифметические и логические выражения.				3	
7	Основные языковые конструкции (условные, циклические, селективные инструкции). Функции: объявление и определение. Передача аргументов в функции. Стандартная библиотека функций языка C++. Библиотека стандартного потокового ввода/вывода. Форматированный ввод/вывод. Файловые потоки.				3	

8	Классы. Инкапсуляция. Сокрытие данных и видимость членов класса. Конструктор. Полный конструктор. Конструктор по умолчанию. Конструктор копирования.				3	
9	Деструктор. Полиморфизм. Перегрузка функций. Перегрузка операторов (унарного, бинарного, особые случаи) Наследование. Виртуальные функции и абстрактные базовые классы. Множественное наследование				3	
10	Интерфейс пользователя. Основные понятия. Стандартизация пользовательского интерфейса. Интерфейс типа "ВОПРОС-ОТВЕТ". Интерфейс командной строки. Текстовый интерфейс. Оконный интерфейс. Графический оконный интерфейс. Web- интерфейс. Социальный интерфейс				2	
11	Современный графический пользовательский интерфейс. Взаимодействие пользователя с программами. Графический пользовательский интерфейс и его реализация в операционной системе Windows				3	
12	Основной объект интерфейса: окно и его основные части. Диалоговое окно и стандартные элементы управления, предназначенные для ввода информации и управления работой программы. Визуализация научных и инженерных данных				3	
13	Уровни абстракции в процессе разработки программного обеспечения: архитектура, структура, реализация). Цикл разработки прикладного программного обеспечения: концептуализация, анализ, проектирование, кодирование, тестирование, эволюция, сопровождение				4	
14	Критерии оценки качества программы. Средства и инструменты разработки программного обеспечения. Стиль программирования. Организация разработки программного обеспечения.				4	

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ВОПРОСОВ

2.1 Цели использования компьютеров при решении прикладных задач. Задачи и особенности прикладного программирования

При изучении вопроса необходимо обратить внимание на следующие особенности.

Роль компьютеров при решении прикладных задач. Задачи и особенности прикладного программирования. Основные инструменты прикладного программиста. Язык программирования - главный инструмент прикладного программиста. Выбор языка программирования. Технологии прикладного программирования: цели, задачи и основные принципы и инструменты. Алгоритмическая и объектно-ориентированная декомпозиция. Принципы объектно-ориентированного анализа: абстрагирование, инкапсуляция, наследование, полиморфизм, модульность, сохраняемость, параллелизм. Объекты и типы объектов. Атрибуты и типы атрибутов. Экземпляры и состояния. Жизненный цикл и поведение объектов: сообщения, события, методы, действия

2.2 Основные инструменты прикладного программиста. Язык программирования - главный инструмент прикладного программиста. Выбор языка программирования

При изучении вопроса необходимо обратить внимание на следующие особенности.

Для того чтобы упростить написание компьютерной программы создан целый набор инструментов.. Каждый инструмент программирования создается с определенной целью.

Для написания программы вам потребуется два очень важных инструмента:
Редактор – чтобы вы смогли написать инструкции для компилятора.

- *Компилятор* – который преобразует ваши инструкции в машинные коды (напоминаем – ЭВМ понимает только машинный язык). Вместо компилятора во многих языках программирования используется *интерпретатор.* Основное различие между ними состоит в том, что интерпретатор преобразует инструкции в машинные коды при каждом запуске программы, а компилятор делает это всего один раз, после чего сохраняет полученный результат в виде выполняемого файла с расширением . exe.
- *Отладчик* – программа, позволяющая найти ошибки и проблемы в работе программы.
- *Программа установки* – установит и настроит работу вашей программы.
- *Программа для создания файлов справки* – позволит вашей программе отображать справочные сведения на экране, а вас избавит от необходимости отображать инструкции на бумаге.

С помощью языка программирования создается не готовая программа, а только ее текст, описывающий ранее разработанный алгоритм. Чтобы получить работающую программу, надо этот текст либо автоматически перевести в машинный код (для этого служат программы-*компиляторы*) и затем использовать отдельно от исходного текста, либо

сразу выполнять команды языка, указанные в тексте программы (этим занимаются *программы-интерпретаторы).*

Языки программирования можно условно разделить на классы.

Другой классификацией языков программирования является их деление на языки, ориентированные на реализацию основ *структурного программирования* и *объектно-ориентированные языки*

Обратить внимание на потенциальные причины использования при написании программ того или иного языка программирования

2.3. Технологии прикладного программирования: цели, задачи и основные принципы и инструменты. Алгоритмическая и объектно-ориентированная декомпозиция. Принципы объектно-ориентированного анализа: абстрагирование, инкапсуляция, наследование, полиморфизм, модульность, сохраняемость, параллелизм. Объекты и типы объектов. Атрибуты и типы атрибутов.

При изучении вопроса необходимо обратить внимание на следующие особенности

Главным показателем технологии являются: содержание отдельных технологических этапов, в частности, способ расчленения всего объема работ на части; принципы распределения отдельных работ между членами коллектива; способ объединения отдельных частей разрабатываемой программы в единое целое и др.

В настоящее время сложились следующие виды технологий:

- процедурное программирование,
- модульное программирование,
- программирование "сверху-вниз",
- структурное программирование,
- HIPO - технология,
- R - технология.

Сущность процедурного программирования заключается в первоначальной разработке отдельных частей программы, из которых собирается программа более высокого иерархического уровня. Отсюда ее второе название - "снизу-вверх".

При разработке сложной программной системы необходимо разделять ее на более простые подсистемы, каждую из которых можно проектировать и программировать самостоятельно, если необходимо также подвергая разделению на части. Такой процесс называется *декомпозицией*. Существует (в первом приближении) два вида декомпозиции:

- алгоритмическая или структурная;
- объектно-ориентированная (далее ОО).

Абстра́кция — в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня. Такой подход является основой объектно-ориентированного программирования. Это позволяет работать с объектами, не вдаваясь в особенности их реализации. В каждом конкретном случае применяется тот или иной подход: инкапсуляция, полиморфизм

или наследование. **Абстракция данных** — популярная и в общем неверно определяемая техника программирования. Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик существенных для корректного ее использования.

Инкапсуля́ция - свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента , а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта. Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: public. Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: private, protected, internal.

Инкапсуляция - один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, полиморфизмом и наследованием). Сокрытие реализации целесообразно применять в следующих случаях: предельная локализация изменений при необходимости таких изменений, прогнозируемость изменений (какие изменения в коде надо сделать для заданного изменения функциональности) и прогнозируемость последствий изменений.

Насле́дование - один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса. Простое наследование: Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class). В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++.В языках, которые позиционируются как наследники C++ (Java, C# и др.), от множественного наследования было решено отказаться в пользу интерфейсов. Большинство современных объектно-ориентированных языков программирования (C#, Java, Delphi и др.) поддерживают возможность одновременно наследоваться от класса-предка и реализовать методы нескольких интерфейсов одним и тем же классом. Этот механизм позволяет во многом заменить множественное наследование — методы интерфейсов необходимо переопределять явно, что исключает ошибки при наследовании функциональности одинаковых методов различных классов-предков.

Полиморфи́зм - возможность объектов с одинаковой спецификацией иметь различную реализацию. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций». Полиморфизм

- один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, инкапсуляцией и наследованием). Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному - интерфейс, класс. Общность имеет внешнее и внутреннее выражение: внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством); внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

2.4. Экземпляры и состояния. Жизненный цикл и поведение объектов: сообщения, события, методы, действия

При изучении вопроса необходимо обратить внимание на следующие особенности.

Объект является экземпляром, который структурирован и функционирует в соответствии со своим классом. Все объекты, порожденные одним и тем же классом, структурированы одним способом, хотя каждый из них имеет свой собственный набор связей атрибутов. Каждая связь атрибута имеет ссылку на экземпляр, обычно на значение данных. Объект может порождаться несколькими классами. В этом случае объект обладает всеми свойствами, которые объявлены во всех этих классах, как структурными, так и поведенческими. К объекту можно добавлять новые классы, а старые классы отделять. Это значит, что свойства новых классов динамически добавляются к данному объекту, а свойства, объявленные ранее в классе, удаляемые из объекта, динамически также удаляются из объекта.

Во всех стандартных моделях, выделяют следующие основные этапы жизненного цикла:

- стратегическое планирование;
- анализ требований;
- проектирование (предварительное и детальное);
- кодирование (программирование);
- тестирование и отладка;
- эксплуатация и сопровождение.

Каждому этапу соответствуют определенный результат и набор документации, являющейся исходными данными для следующего этапа. В заключение каждого этапа производится верификация документов и решений с целью проверки их соответствия первоначальным требованиям заказчика

2.5. Структура программы на языке C++. Проект. Компиляция программы и сборка исполняемого модуля. Размещение программы и данных в памяти. Структура исполняемого модуля. Переменные: объявление, определение, инициализация. Переменные: значение, указатель, ссылка. Время жизни, области видимости и классы памяти переменных

При изучении вопроса необходимо обратить внимание на следующие особенности.

Структура программы. Общая структура программы на Си/Си++ следующая:
директивы_препроцессора

определение_функции_1
определение_функции_2
определение_функции_N

Среди функций обязательно присутствует главная функция с именем main. Простейшая программа содержит только главную функцию и имеет следующую структуру:

директивы препроцессора
void main()
{ определения_объектов;
исполняемые_операторы;
}

Понятие «оператор» в Си трактуется следующим образом: любое выражение, после которого стоит точка с запятой, воспринимается компилятором как отдельный оператор. Оператор определяет законченное действие на очередном шаге выполнения программы.

Объединенная единым алгоритмом совокупность описаний и операторов образует программу на алгоритмическом языке. Для того чтобы выполнить программу, требуется перевести ее на язык, понятный процессору — в машинные коды.

Сначала программа передается препроцессору, который выполняет директивы, содержащиеся в ее тексте (например, включение в текст так называемых заголовочных файлов - текстовых файлов, в которых содержатся описания используемых в программе элементов.

Получившийся полный текст программы поступает на вход компилятора, который выделяет лексемы, а затем на основе грамматики языка распознает выражения и операторы, построенные из этих лексем. При этом компилятор выявляет синтаксические ошибки и в случае их отсутствия строит объектный модуль.

Компоновщик, или редактор связей, формирует исполняемый модуль программы, подключая к объектному модулю другие объектные модули, в том числе содержащие функции библиотек, обращение к которым содержится в любой программе (например, для осуществления вывода на экран). Если программа состоит из нескольких исходных файлов, они компилируются по отдельности и объединяются на этапе компоновки. Исполняемый модуль имеет расширение .exe и запускается на выполнение обычным образом.

Все данные, хранящиеся в памяти компьютера, представляются в двоичной системе счисления в виде машинных слов – наборов двоичных разрядов (совокупность битов – нулей и единиц). Биты объединяются в последовательности: байты, слова и т.д. Минимальный размер машинного слова для хранения, обработки и передачи данных – 8 битов (1 байт). Все остальные слова кратны ему. Стандартное машинное слово имеет разрядность, соответствующую разрядности процессора (32 разряда или 4 байта).

Любой алгоритм можно сформулировать в виде последовательности операций, выполняемых над некоторыми объектами, и записать эту последовательность операций в виде программы. С увеличением алгоритмов и соответственно объема программы невозможно удержать в памяти все ее детали. Возрастает трудоемкость создания таких программ и уменьшается их надежность. К счастью программисты давно используют принципы структурного программирования, разбивая программу на отдельные процедуры, реализующие определенные группы операции алгоритма. Каждая процедура решает свою узкоспециализированную задачу, может обращаться к другим процедурам, а для этого ей необходимо уметь обмениваться данными с другими процедурами. Понятие подобной процедуры в языке С++ конкретизируется в виде объекта типа функции.

Переменные обычно объявляются в начале программы. Для объявления надо написать название типа переменных (int, float или char), а затем через запятую имена всех объявляемых переменных.

Кроме обычных переменных и указателей в Си++ имеется тип "ссылка на переменную", задающий для переменной дополнительное имя (псевдоним). Внутреннее представление ссылки такое же, как указателя, т.е. в виде адреса переменной, но обращение к переменной по ссылке записывается в той же форме, что и обращение по основному имени. Переменная типа ссылки всегда инициализируется заданием имени переменной, к которой относится ссылка. При объявлении ссылки за именем типа записывается знак & (амперсенд):

int ii;int& aii = ii;

Время жизни - это интервал времени выполнения программы, в течение которого программный объект (переменная или функция) существует. Время жизни переменной может быть локальным или глобальным. Переменная с глобальным временем жизни имеет распределенную для нее память и определенное значение на протяжении всего времени выполнения программы, начиная с момента выполнения объявления этой переменной. Переменная с локальным временем жизни имеет распределенную для него память и определенное значение только во время выполнения блока, в котором эта переменная определена или объявлена. При каждом входе в блок для локальной переменной распределяется новая память, которая освобождается при выходе из блока.

Все функции в СИ имеют глобальное время жизни и существуют в течение всего времени выполнения программы.

Область видимости - это часть текста программы, в которой может быть использован данный объект. Объект считается видимым в блоке или в исходном файле, если в этом блоке или файле известны имя и тип объекта. Объект может быть видимым в пределах блока, исходного файла или во всех исходных файлах, образующих программу. Это зависит от того, на каком уровне объявлен объект: на внутреннем, т.е. внутри некоторого блока, или на внешнем, т.е. вне всех блоков.

2.6. Динамическое размещение данных в памяти. Составные типы данных. Массивы - как пример гомогенной структуры данных: размещение в памяти, доступ к элементам. Одномерные и многомерные массивы. Структуры - как пример гетерогенной структуры данных. Реализация вычислительных операций. Арифметические и логические выражения

При изучении вопроса необходимо обратить внимание на следующие особенности.

Применяются три уровня структур данных:

1. содержательный,
2. логический,
3. физический.

На содержательном уровне структур данных исследуются конкретные объекты обработки, их свойства и отношения между объектами. На этом уровне важны не только значения, но и смысл данных.

На логическом или абстрактном (логические структуры) уровне структура данных считается машинно-независимым логическим понятием, и выделяются следующие задачи: определение массивов данных как объектов исследования, выделение состава массива, определение структуры данных по заданным требованиям, разработка количественных методов оценки эффективности различных видов структур данных.

Физический уровень (физическая структура). На этом уровне рассматриваются память ЭВМ и представление в ней значений (ячейки, разряды ячеек, их адреса и взаимное расположение значений.), т.е. отображение данных в памяти ЭВМ

По уровню сложности структуры данных разделяются на

1. **простые** – обычные переменные или константы языков программирования стандартных типов, а также динамические переменные этих же типов;

2. **наборы однотипных данных** – массивы, одно- и многомерные;

3. **интегральные** – записи и объекты классов и подобные структуры;

4. **динамические структуры с внутренними связями** – списки, деревья, графы.

По способу создания структуры данных можно разделить на

1. **обычные** – переменные стандартных типов, обычные (т.е. не динамические) массивы, записи и т.п.;

2. **динамические** (создаваемые и разрушаемые с помощью специальных операций или процедур динамического выделения и освобождения памяти) – динамические массивы, динамические переменные, связные списки, деревья.

С точки зрения **архитектуры** можно выделить:

1. **линейные** структуры – одномерные массивы, линейные списки, линейные очереди, стеки;

2. **прямоугольные** структуры – двумерные и многомерные массивы;

3. **кольцевые** структуры – кольцевые списки, кольцевые очереди, некоторые реализации графов;

ветвящиеся структуры – деревья различных видов, некоторые реализации графов;

5. **сетевые** структуры – графы.

В зависимости от **наличия** или отсутствия **связей** различают:

1. **несвязные** структуры – векторы, массивы, строки, стеки, очереди;

2. **связные** – списки, деревья, графы.

В зависимости от **постоянства** во время работы программы различают:

1. **статические** (неизменяющиеся) структуры – переменные различных типов, записи, массивы, в том числе динамические.

2. **динамические** (изменяющиеся) – списки, деревья, очереди, стеки, в общем случае графы.

По месту размещения:

1. существующие в памяти ЭВМ (или **внутренние**) – ими могут быть все ранее рассмотренные примеры структур данных;

2. хранящиеся на **внешних** носителях (внешних ЗУ) – файлы и системы файлов

Массивы - это группа элементов одинакового типа (double, float, int и т.п.). Из объявления массива компилятор должен получить информацию о типе элементов массива и их количестве. Объявление массива имеет два формата:

спецификатор-типа описатель [константное - выражение];

спецификатор-типа описатель [ ];

Описатель - это идентификатор массива .

Спецификатор-типа задает тип элементов объявляемого массива. Элементами массива не могут быть функции и элементы типа void.

Константное-выражение в квадратных скобках задает количество элементов массива.

Массив - это непрерывный участок памяти, содержащий последовательность объектов одинакового типа, обозначаемый одним именем.

Массив характеризуется следующими основными понятиями:

Элемент массива (значение элемента массива) – значение, хранящееся в определенной ячейке памяти, расположенной в пределах массива, а также адрес этой ячейки памяти.

Каждый элемент массива характеризуется тремя величинами:

- адресом элемента - адресом начальной ячейки памяти, в которой расположен этот элемент;
- индексом элемента (порядковым номером элемента в массиве);
- значением элемента.

Адрес массива – адрес начального элемента массива.

Имя массива – идентификатор, используемый для обращения к элементам массива.

Размер массива – количество элементов массива

Размер элемента – количество байт, занимаемых одним элементом массива.

В языке Си могут быть также объявлены многомерные массивы. Отличие многомерного массива от одномерного состоит в том, что в одномерном массиве положение элемента определяется одним индексом, а в многомерном — несколькими. Примером многомерного массива является матрица.

Общая форма объявления многомерного массива

тип имя[размерность1][размерность2]...[размерность m];

Структуры - это составной объект, в который входят элементы любых типов, за исключением функций. В отличие от массива, который является однородным объектом, структура может быть неоднородной. Тип структуры определяется записью вида:

struct { список определений }

Основными арифметическими операциями являются: сложение (‘+’), вычитание (‘-‘), умножение (‘*’) и деление (‘/’). Порядок выполнения операций в выражении соответствует их приоритету. Операции с одинаковым приоритетом в выражении выполняются слева направо.

Операция деления (‘/’) выполняется согласно типу ее операндов. Если оба операнда являются целыми числами, то деление будет целочисленным. Если один из операндов является вещественным, то и результат будет вещественным.

Операция вычисления остатка в Си обозначается символом ‘%’

2.7. Основные языковые конструкции (условные, циклические, селективные инструкции). Функции: объявление и определение. Передача аргументов в функции. Стандартная библиотека функций языка С++. Библиотека стандартного потокового ввода/вывода. Форматированный ввод/вывод. Файловые потоки.

При изучении вопроса необходимо обратить внимание на следующие особенности

Программа, написанная на языке Си, состоит из операторов. Каждый оператор вызывает выполнение некоторых действий на соответствующем шаге выполнения программы.

При написании операторов применяются латинские прописные и строчные буквы, цифры и специальные знаки. К таким знакам, например, относятся: точка (.), запятая (,), двоеточие (:), точка с запятой (;) и др. Совокупность символов, используемых в языке, называется алфавитом языка.

В персональном компьютере символы хранятся в виде кодов. Соответствие между каждым символом и его кодом задается специальной кодовой таблицей. На нее разработан стандарт ASCII, поэтому коды символов называют ASCII-кодами.

Различают видимые и управляющие символы. Первые могут быть отображены на экране дисплея либо отпечатаны на принтере. Вторые вызывают определенные действия в машине,

Для представления каждого символа в персональном компьютере используется один байт, поэтому общее число символов равно $2^8 = 256$.

Важным понятием языка является идентификатор, который используется в качестве имени объекта (функции, переменной, константы и др.).

В программах на языке Си важная роль отводится комментариям. Они повышают наглядность и удобство чтения программ

Программы на языке Си обычно состоят из большого числа отдельных функций (подпрограмм). Как правило, эти функции имеют небольшие размеры и могут находиться как в одном, так и в нескольких файлах. Все функции являются глобальными. В языке запрещено определять одну функцию внутри другой. Связь между функциями осуществляется через аргументы, возвращаемые значения и внешние переменные.

В общем случае функции в языке Си необходимо объявлять. Объявление функции (т.е. описание заголовка) должно предшествовать ее использованию, а определение функции (т.е. полное описание) может быть помещено как после тела программы (т.е. функции main( )), так и до него. Если функция определена до тела программы, а также до ее вызовов из определений других функций, то объявление может отсутствовать. Как уже отмечалось, описание заголовка функции обычно называют прототипом функции.

Функция объявляется следующим образом:

тип имя_функции(тип имя_параметра_1, тип имя_параметра_2, ...);

Тип функции определяет тип значения, которое возвращает функция. Если тип не указан, то предполагается, что функция возвращает целое значение (int).

В программы на языке Си можно передавать некоторые аргументы. Когда вначале вычислений производится обращение к main( ), ей передаются три параметра. Первый из них определяет число командных аргументов при обращении к программе. Второй представляет собой массив указателей на символьные строки, содержащие эти аргументы (в одной строке - один аргумент). Третий тоже является массивом указателей на символьные строки, он используется для доступа к параметрам операционной системы (к переменным окружения).

В системах программирования подпрограммы для решения часто встречающихся задач объединяются в библиотеки. К числу таких задач относятся: вычисление математических функций, ввод/вывод данных, обработка строк, взаимодействие со средствами операционной системы и др. Использование библиотечных подпрограмм избавляет пользователя от необходимости разработки соответствующих средств и предоставляет ему дополнительный сервис. Включенные в библиотеки функции поставляются вместе с системой программирования. Их объявления даны в файлах *.h (это так называемые включаемые или заголовочные файлы).

Язык программирования СИ+ поддерживает множество функций стандартных библиотек для файлового ввода и вывода. Эти функции составляют основу заголовочного файла стандартной библиотеки языка Си <stdio.h>.

Функциональность ввода-вывода языка Си по текущим стандартам реализуется на низком уровне. Язык Си абстрагирует все файловые операции, превращая их в операции с потоками байтов, которые могут быть как «потоками ввода», так и «потоками вывода». В отличие от некоторых ранних языков программирования, язык Си не имеет прямой поддержки произвольного доступа к файлам данных; чтобы считать записанную информацию в середине файла, программисту приходится создавать поток, ищущий в середине файла, а затем последовательно считывать байты из потока

В языке Си отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.д. Эти устройства сильно отличаются друг от друга. Однако файловая

система преобразует их в единое абстрактное логическое устройство, называемое потоком.

В Си существует два типа потоков: текстовые (text) и двоичные (binary).

Текстовый поток - это последовательность символов. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки).

Двоичный поток - это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

Прежде чем читать или записывать информацию в файл, он должен быть открыт и тем самым связан с потоком. Это можно сделать с помощью библиотечной функции fopen( )

2.8. Классы. Инкапсуляция. Сокрытие данных и видимость членов класса. Конструктор. Полный конструктор. Конструктор по умолчанию. Конструктор копирования.

При изучении вопроса необходимо обратить внимание на следующие особенности

Основным элементом конструкции в объектно-ориентированных языках служит модуль, составленный из логически связанных классов и объектов, а не подпрограмма, как в процедурно-ориентированных языках.

Объектно-ориентированное программирование – это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса, а классы образуют иерархию на принципах наследуемости.

В объектно-ориентированных языках сокращена или отсутствует область глобальных данных. Данные и действия организуются таким образом, что основой программы становятся классы и объекты. Объектно-ориентированное программирование (ООП) позволяет программисту разложить проблему на связанные между собой подзадачи. Каждая проблема становится самостоятельным объектом, содержащим свои собственные коды и данные, которые относятся к этому объекту. В этом случае задача программирования упрощается и программист получает возможность оперировать с гораздо большими по объему программами. ООП снижает стоимость разработки надежных программ.

Основные принципы объектно-ориентированного программирования. Основными принципами ООП являются: абстракция данных, инкапсуляция, полиморфизм, наследование.

Абстракция данных означает возможность создания новых типов данных на базе существующих. В C++ поддерживается посредством определения классов.

Инкапсуляция (или сокрытие информации) – это механизм, который объединяет данные и код, манипулирующий с этими данными, и защищает и то и другое от внешнего вмешательства или неправильного использования. В ООП код и данные объединяются вместе, можно сказать, что создается так называемый “черный ящик”. Все необходимые данные и коды находятся внутри него. Так как ранее было показано, что объект содержит в себе как данные, так и операции над ними, то можно сказать, что объект –это то, что поддерживает инкапсуляцию.

Полиморфизм - возможность определить внутреннюю процедуру, исходя из типа данных. В более общем смысле концепцией полиморфизма является идея “один интерфейс, множество методов”. Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. При этом выполнение конкретного действия зависит от данных. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компиля-

тор. Иллюстрацией применения полиморфизма, в программе является то, что независимо от типа данных ввод и вывод поддерживаются единообразно.

Наследование – это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование позволяет поддерживать концепцию иерархии классов и играет очень важную роль в ООП.

Совокупность принципов проектирования, разработки и реализации программ, которая базируется на абстракции данных, предусматривает создание новых типов данных, с наибольшей полнотой отображающих особенности решаемой задачи. Одновременно с данными для каждого типа вводится набор операций, удобных для обработки этих данных. В языке С++ программист имеет возможность вводить собственные типы данных и определять операции над ними с помощью классов.

Класс фундаментальное понятие С++, с введением которого язык С++ становится объектно-ориентированным языком. Класс – это тип, создаваемый программистом на основе уже существующих типов.

Класс-это именованный набор компонентов, включающий структуры данных и связанные с ними функции их обработки.

Таким объявлением класса определен не просто отдельный объект – трехмерный вектор, а любой произвольный вектор. Здесь задан целый класс объектов.

В отличие от обычного определения данных при описании элементов класса не допускается их инициализация. Это естественное свойство класса, так как при его определении еще не существует участков памяти, соответствующих его элементам-данным. Память выделяется не для класса, а только для объектов класса. Для инициализации элементов-данных объектов должны использоваться специальные функции, называемые конструкторами.

Служебное слово public делит тело класса на две части. Первая часть, до слова public по умолчанию является закрытой. Это означает, что имена в этой части – x, y, z могут использоваться только функциями – методами класса. Вторая, открытая часть, составляет интерфейс к объекту класса. Служебное слово public определяет, что метод класса – функция mod( ) является общедоступной, то есть может быть вызвана из любого места программы.

В свойстве класса ограничивать доступ к переменным, объявленным внутри него, реализуется такой принцип ООП, как инкапсуляция. Что дает инкапсуляция программисту: внутреннюю реализацию класса можно изменять, не опасаясь, что это повлечет нежелательные побочные эффекты в остальной части программы. Инкапсуляция позволяет уменьшить время отладки программы за счет локализации участка программы, содержащего ошибку.

Данные, описанные внутри класса, как правило, имеют атрибут доступа private. Это означает, что данные класса доступны только методам этого класса. В таком случае следует предусмотреть функцию –метод класса для инициализации данных.

Функция создания и инициализации объектов данного класса называется конструктором.

Конструкторы обладают большинством характеристик обычных методов – их следует объявить и определить в пределах класса, либо объявить в классе, но определить вне его. Однако они обладают некоторыми уникальными свойствами:

конструкторы имеют то же имя, что и класс;

не имеют объявлений возврата (даже void);

не могут быть унаследованы, хотя производный класс может вызывать конструкторы базового класса;

вызвать конструктор тем же образом, что и обычную функцию – метод класса, нельзя. Конструктор вызывается неявно, при создании или копировании объекта данного класса.

Можно сказать, что конструктор превращает фрагмент памяти в объект того типа, который предусмотрен определением класса.

Конструкторы глобальных переменных вызываются до вызова функции main(). Локальные объекты создаются, как только становится активной сфера действия переменной. С помощью параметров конструктору могут быть переданы любые данные, необходимые для создания и инициализации объектов. Параметры конструктора могут быть любого типа, за исключением класса, элементом которого является данный конструктор

Такой конструктор называется конструктором, получающим параметры. Конструкторы могут иметь аргументы по умолчанию.

Конструктор по умолчанию не принимает аргументов вообще, его не следует путать с конструктором, имеющим аргументы по умолчанию.

Конструктор копирования для класса . X - это такой конструктор, который может быть вызван с единственным аргументом типа X:

X:: X(const X&)

Конструктор копирования запускается при копировании объекта данного класса, обычно в случае объявления с инициализацией объектом другого класса: X x = y. Если конструкторы в теле класса не были определены явно, то компилятор формирует по умолчанию конструктор без параметров и конструктор копирования.

Перегрузка конструкторов В определении класса может присутствовать несколько конструкторов, т.е. конструкторы могут быть перегружены. Перегруженные конструкторы позволяют создавать объекты в зависимости от значений, используемых при инициализации. Перегрузка конструкторов иллюстрирует такой принцип ООП, как полиморфизм, что означает наличие в пределах одного класса нескольких функций с одним и тем же именем

2.9. Деструктор. Полиморфизм. Перегрузка функций. Перегрузка операторов (унарного, бинарного, особые случаи)

Наследование. Виртуальные функции и абстрактные базовые классы. Множественное наследование

При изучении вопроса необходимо обратить внимание на следующие особенности

Деструктор - это функция уничтожения объекта данного класса. Деструктор представляет метод с именем, совпадающим с именем класса, перед которым стоит символ ~(тильда). Это универсальный символ, обозначающий “не” (“не конструктор”).

Деструктор не должен иметь параметров, и, как и конструктор - типа возврата.

Класс может иметь только один деструктор или не иметь вообще. Если деструктор не объявлен для класса явно, компилятор генерирует его автоматически.

Вызов деструктора выполняется неявно, когда объект выходит из своей объявленной сферы действия.

Когда указатели объектов выходят за предел сферы действия, неявный вызов деструктора не происходит. Для уничтожения таких указателей должна быть явно задана операция delete.

Перегрузка операций выполняется с помощью операций - функций. Синтаксис объявления прототипа операции – функции: Тип_возвращаемого значения

operator знак_операции

(список параметров операции-функции);

Можно перегрузить унарные операции, такие, как ++ или --. При перегрузке унарных операций с помощью функций – методов класса функция-оператор не имеет параметров. Унарная операция выполняется над объектом, осуществляющим вызов функции-оператора путем неявной передачи указателя this.

Для перегрузки операций следует пользоваться функциями-методами. Функции-друзья предназначаются в С++ большей частью для специальных ситуаций.

Особенности использования дружественных функций

Дружественная функция при вызове не получает указателя this. Объекты классов должны передаваться дружественной функции только явно через аппарат параметров.

Такая перегрузка операций позволяет программисту организовывать ввод типов данных базовых типов единообразно. В зависимости от типа данного компилятор связывает вызов функции-оператора с соответствующим определением.

Перегрузка операций >> и << для ввода, вывода типов, объявленных пользователем
Очень полезным примером применения механизма перегрузки операций и функций-друзей классов является использование операций >> и << для ввода, вывода типов, объявленных пользователем.

Следует обратить внимание на то, что в качестве возвращаемых значений функций, перегружающих операции >> и << объявлены ссылки на потоки ввода и вывода соответственно. Это необходимо, чтобы в одном операторе можно было записать несколько операций извлечения и вставки. Далее, функции имеют два параметра. Первым служит ссылка на поток, который присутствует в левой части операций >> и <<, а вторым параметром служит объект с правой стороны этих операций. Перегруженные функции operator >> и operator << это функции-друзья класса. Действительно, ни функция вставки, ни функция извлечения не могут быть методами класса. Причина заключается в том, что если функция-оператор является методом класса, то левым операндом, неявно передаваемым с использованием указателя this служит объект того класса, который осуществляет вызов функции-оператора. Левым же аргументом функций операторов, перегружающих операции >> и <<является ссылка на потоки ввода или вывода, которые не принадлежат классу, создаваемому пользователем в программе. Поэтому перегруженные функции операторы operator >> и operator << не могут быть методами класса, созданного пользователем. Это позволяет сохранить неизменным принцип инкапсуляции ООП.

Язык С++ позволяет классу наследовать элементы данных и элементы-функции одного или нескольких других классов. Другими словами, новый класс может получить атрибуты и поведение от уже существующего класса.

Таким образом создается иерархия типов данных (классов). Имеющиеся классы – называют базовыми, а новые – производными (наследниками). Производные классы получают ,,наследство,, -данные и методы своих базовых классов и пополняются собственными данными и методами. В свою очередь, производный класс может служить базовым для другого класса.

Наследование позволяет абстрагировать некоторое общее или схожее поведение различных объектов в одном базовом классе.

Производный класс может иметь не один, а множество базовых классов. Это называется множественным наследованием. Когда наследуется множество базовых классов, конструкторы выполняются слева направо в порядке, задаваемом в объявлении производного класса. Деструкторы выполняются в обратном порядке.

К механизму виртуальных функций (virtual function) обращаются в тех случаях, когда в базовый класс необходимо поместить функцию, которая должна по-разному выполняться в производных классах. Виртуальные функции важны потому, что они использу-

ются для поддержки динамического полиморфизма. В С++ такое свойство ООП, как полиморфизм, поддерживается двумя способами:

на этапе компиляции - посредством перегрузки операций и функций;

во время выполнения программы – посредством виртуальных функций.

Виртуальные функции позволяют производным классам определить разные версии функций базового класса.

Класс, содержащий одну или несколько производных функций, называется полиморфным классом.

При использовании механизма виртуальных функций нельзя изменять тип возвращаемого значения и список аргументов функции. Если две функции в базовом и производном классах с одним и тем же именем имеют разные аргументы и разные типы возвращаемых значений, причем в базовом классе функция объявлена как виртуальная, то механизм виртуальных функций игнорируется.

Чисто виртуальные функции и абстрактные классы

Виртуальная функция, объявленная в базовом классе, часто никогда не используется для объектов базового класса, т.е. не имеет определения.

Версия виртуальной функции, которая, с одной стороны, должна быть определена, а с другой, никогда не должна использоваться, должна быть объявлена как чисто виртуальная функция.

При выполнении программы при обращении к чисто виртуальной функции выдается соответствующее сообщение и программа аварийно завершается.

Чисто виртуальные функции полезны в следующем: они позволяют установить контроль со стороны компилятора за ошибочным созданием объектов “фиктивных” типов

Класс с одной или большим количеством чисто виртуальных функций называется абстрактным классом.

Правила языка С++ запрещают создание объектов таких типов. Абстрактный класс может быть использован только как базовый для последующих порождений новых классов.

Причина, по которой абстрактные классы не могут использоваться как объявления объекта, состоит в том, что одна или более функция не имеет определения.

Шаблоны позволяют создать универсальный фрагмент кода, а затем использовать его многократно с различными типами данных или различными объектами. С помощью шаблонов можно уменьшить размер и сложность программ.

Шаблоны функций позволяют использовать в качестве аргумента функций тип переменной.

2.10. Интерфейс пользователя. Основные понятия. Стандартизация пользовательского интерфейса. Интерфейс типа "ВОПРОС- ОТВЕТ". Интерфейс командной строки. Текстовый интерфейс. Оконный интерфейс. Графический оконный интерфейс. Web- интерфейс. Социальный интерфейс

При изучении вопроса необходимо обратить внимание на следующие особенности

Доступные пользователю способы взаимодействия с программами и устройствами компьютера называют пользовательским интерфейсом.

Пользовательский интерфейс — совокупность способов организации диалога «человек — компьютер». Он включает возможности задания пользователем команд, например запуска программы на выполнение; виды и способы вывода сообщений компьютера в ответ на команды пользователя; виды сообщений о состоянии устройств и т. д.

Одним из самых старых является интерфейс командной строки (консоль) — разновидность текстового интерфейса. При таком способе взаимодействия пользователь вводит (в основном с клавиатуры) в специальную командную строку текстовые команды, являющиеся инструкциями, понятными операционной системе.

В настоящее время практически все пользователи компьютеров ведут диалог с операционной системой при помощи графического интерфейса.

В совокупности данные инструменты образуют интерфейс программы - внешний вид отдельных её элементов и видов на экране компьютера. Поскольку в различных программах используется много однотипных ситуаций и вариантов взаимодействия пользователей с программами, возникает потребность стандартизировать их интерфейсы.

Интерфейс (Interface) в широком смысле - это определённая стандартами граница между взаимодействующими независимыми объектами. Интерфейс задаёт параметры, процедуры и характеристики взаимодействия объектов.

Интерфейс определяет:

-язык пользователя;

-язык сообщений компьютера, организующий диалог на экране дисплея;

-знания пользователя.

Язык пользователя - это те действия, которые пользователь производит при работе с системой путём использования возможностей клавиатуры, пишущих на экране электронных карандашей, джойстика, мыши, подаваемых голосом команд и т.п. Наиболее простой формой языка пользователя является создание форм входных и выходных документов. Получив входную форму (документ), пользователь заполняет его необходимыми данными и вводит в компьютер. Система поддержки принятия решений производит необходимый анализ и выдает результаты в виде выходного документа установленной формы.

Язык сообщений - это то, что пользователь видит на экране дисплея (символы, графика, цвет); это данные, полученные на принтере; звуковые выходные сигналы и т.п. Важным измерителем эффективности используемого интерфейса является выбранная форма диалога между пользователем и системой. Наиболее распространены следующие формы диалога: запросно-ответный режим, командный режим, режим меню, режим заполнения пропусков в выражениях, предлагаемых компьютером.

Каждая форма в зависимости от типа задачи, особенностей пользователя и принимаемого решения имеет достоинства и недостатки. Долгое время единственной реализацией языка сообщений был отпечатанный или выведенный на экран дисплея отчёт или сообщение. Теперь представление выходных данных осуществляется с помощью машинной графики. Она позволяет создавать на экране и бумаге цветные графические изображения в двумерном и трёхмерном виде. Использование машинной графики, значительно повышает наглядность и интерпретацию выходных данных, всё чаще используется в информационных технологиях поддержки принятия решений. Командный интерфейс - самый простой. Он обеспечивает выдачу на экран системного приглашения для ввода команды.

Некогда ранее распространенный командный интерфейс имеет ряд существенных недостатков с точки зрения пользователя: многочисленность команд, отсутствие стандарта для приложений и т.д.

Появились новые виды интерфейса, такие как биометрический (мимический) и семантический (общественный). В связи с этим поставлена проблема создания общественного интерфейса (social interface). Общественный интерфейс будет включать в себя лучшие решения WIMP- и SILK-интерфейсов.

Предполагается, что при использовании общественного интерфейса не нужно будет разбираться в меню. Экранные образы однозначно укажут дальнейший путь. Перемещение от одних поисковых образов к другим будет проходить по смысловым семантическим связям [12]. Современный интерфейс пользователя - графический интерфейс. Устройства графического ввода/вывода выполняют функции обеспечения интерфейсного диалога компьютера с человеком при вводе команд и запросов в систему, а также функции

обеспечения выполнения информационных процессов. Пользователю достаточно помнить минимальное количество информации командного, процессуального характера, чтобы иметь возможность оперативно принимать соответствующие решения. Для этого ему необходимо владеть алгоритмами функционирования подсистемы «человек-техническое средство» и профессиональными навыками взаимодействия с ЭВМ.

Процедурно-ориентированный интерфейс использует традиционную модель взаимодействия с пользователем, основанную на понятиях «процедура» и «операция». В рамках этой модели программное обеспечение предоставляет пользователю возможность выполнения некоторых действий, для которых пользователь определяет соответствие данных и следствием выполнения которых является получение желаемого результата.

Объектно-ориентированные интерфейсы используют модель взаимодействия с пользователем, ориентированную на манипулирование объектами предметной области. В рамках этой модели пользователю предоставляется возможность напрямую взаимодействовать с каждым объектом и инициировать выполнение операций, в процессе которых взаимодействуют несколько объектов. Задача пользователя формулируется как целенаправленное изменение некоторого объекта. Объект понимается в широком смысле слова - модель БД, системы и т.д. Объектно-ориентированный интерфейс предполагает, что взаимодействие с пользователем осуществляется посредством выбора и перемещения пиктограмм соответствующей объектно-ориентированной области. Различают однодокументные (SDI) и многодокументные (MDI) интерфейсы.

2.11. Современный графический пользовательский интерфейс. Взаимодействие пользователя с программами. Графический пользовательский интерфейс и его реализация в операционной системе Windows.

При изучении вопроса необходимо обратить внимание на следующие особенности

Графический пользовательский интерфейс— разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений.

В отличие от интерфейса командной строки, в GUI пользователь имеет произвольный доступ (с помощью устройств ввода — клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Чаще всего элементы интерфейса в GUI реализованы на основе метафор и отображают их назначение и свойства, что облегчает понимание и освоение программ неподготовленными пользователями.

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.

Пользовательский графический интерфейс Windows

Термин "интерфейс" широко используется в областях, где человеку приходится иметь дело с обработкой информации на компьютере. В переводе с английского языка Interface означает внешнее лицо. В компьютерном мире известно множество разновидностей интерфейсов: интерфейс пользователя, графический интерфейс, интерфейс ввода-вывода, внешний или внутренний интерфейс, интеллектуальный интерфейс, человеко-машинный интерфейс, программный интерфейс и др.

Интерфейс – совокупность средств и правил, которые обеспечивают взаимодействие устройств, программ и человека.

Особенно важен интерфейс, обеспечивающий взаимодействие пользователя с персональным компьютером, называемый пользовательским интерфейсом. От удобства этого интерфейса во многом зависит успех нового программного продукта в конкурентной

борьбе на рынке программных средств. Пользовательский интерфейс может быть символьным и графическим.

Символьный интерфейс используется обычно при работе видеосистемы в текстовом режиме. Информация выводится на экран монитора посимвольно. До появления Windows все операционные системы, в том числе MS DOS и ее оболочка Norton Commander , предоставляли пользователю символьный интерфейс. Он достаточно экономичен по потреблению ресурсов и способен обеспечить вполне комфортную работу пользователя. Исключение составляет интерфейс командной строки операционной системы MS DOS , который требует от пользователя знания синтаксиса команд. Следует заметить, что символьный интерфейс Norton Commander не вызывает особых трудностей у неквалифицированного пользователя и может использоваться в графическом режиме работы монитора.

Графический интерфейс появляется тогда, когда видеосистема может работать в графическом режиме, т.е. выводить на экран монитора информацию поточечно. Переход к графическому пользовательскому интерфейсу стал возможным благодаря улучшению технических характеристик персонального компьютера. Такой интерфейс предъявляет повышенные требования к быстродействию видеосистемы, но вместе с тем при этом достигается основная цель - создается комфортная среда работы пользователя, так как человеку более естественно и удобно оперировать образами (картинками). Графический интерфейс по сравнению с символьным воспринимается как более понятный и интуитивно ясный.

Графический пользовательский интерфейс – интерфейс, где для взаимодействия человека и компьютера используются графические средства

Ярким примером графического пользовательского интерфейса служит интерфейс Windows . При разработке этой операционной системы специалисты широко использовали возможные графические средства: рисунки, специальные значки, цветовое оформление, разнообразные начертания шрифтов, дизайн экрана и др. В результате интерфейс стал "дружественным" по отношению к человеку и уже не требует специальных программистских знаний, как было раньше в других операционных системах.

Графический интерфейс Windows позволяет более оперативно задавать команды операционной системы, запускать программы, выбирать файлы и параметры, указывая на соответствующие значки, кнопки, пункты меню, элементы списка, флажки и др. Набор используемых элементов интерфейса стандартен, что позволяет после изучения интерфейса Windows легко и быстро осваивать интерфейс приложений Windows

2.12. Основной объект интерфейса: окно и его основные части. Диалоговое окно и стандартные элементы управления, предназначенные для ввода информации и управления работой программы. Визуализация научных и инженерных данных.

При изучении вопроса необходимо обратить внимание на следующие особенности.

Окна - объекты графического интерфейса

Основу нового графического интерфейса пользователя составляет организованная и хорошо продуманная система окон и других графических объектов.

Окно – обрамленная прямоугольная область на экране монитора, в которой отображаются приложение, документ, сообщение. Окно будет активным (текущим), если с ним в данный момент работает пользователь.

Окна на экране монитора (на электронном рабочем столе) аналогичны листам бумаги, располагающимся на столе и содержащим какую-либо информацию. Различают несколько типов окон, вид которых и появление на экране монитора определяются отображаемой в них информацией. Структура этих окон выполнена в соответствии с разработанным стандартом. Такими типовыми окнами, которые будут рассмотрены ниже, являются

окно приложений, окно документа, диалоговое окно. Окно справочной системы, также являющееся типовым, является разновидностью диалогового окна, но в нем дополнительно предусмотрена возможность использования гиперссылок для быстрого перехода к различным разделам справки.

Общая концепция Windows состоит в максимальной стандартизации всех элементов и приемов работы, чтобы при подключении нового приложения не надо было осваивать все заново. Поэтому структура окон максимально унифицирована и пользовательский интерфейс очень однообразен.

Окна приложения и документа

Перечислим стандартные элементы, которые составляют окно любого приложения:

рабочее поле, где располагаются создаваемые в этом приложении документы. Они могут занимать весь экран, а могут быть свернуты в значки;

управляющее (основное) меню, содержащее имена ниспадающих меню;

ниспадающее меню, содержащее группы команд, объединенных по функциональному назначению;

панели инструментов, представляющие собой линейки командных кнопок для быстрого выбора наиболее часто используемых команд;

заголовок окна, в котором отображается название приложения;

кнопка системного меню, с помощью которого вызываются команды изменения размеров окна и его перемещения;

кнопки <Свернуть>, <Развернуть> (<Восстановить>) и <Закрыть>, дублирующие команды системного меню и служащие для ускорения их вызова;

строка состояния, содержащая информацию о режимах работы приложения.

Окно документа всегда встроено в окно приложения. По своей структуре оно напоминает окно приложения, но существенно проще. Окно документа во многих приложениях имеет стандартный вид с элементами:

рабочее поле, где создается документ средствами приложения;

вертикальные и горизонтальные линейки прокрутки. Они появляются в документах, занимающих места больше, чем площадь рабочего поля, и служат для просмотра документа по вертикали и горизонтали;

заголовок окна документа, где отображается его название, совпадающее с именем файла;

кнопка системного меню и кнопки <Свернуть>, <Развернуть> (<Восстановить>) и <Закрыть> имеют то же значение, что и в окне приложения.

Если в данном приложении открыто несколько окон документов, то пользователь может работать лишь в одном из этих окон. Это окно называется активным, его заголовок выделяется цветом. Остальные окна оказываются пассивными. Если окна документов перекрываются, активное окно закрывает все остальные.

Диалоговое окно служит для настройки параметров операционной системы или приложения, а также выводит необходимые в процессе работы сообщения. Оно выводится приложением или операционной системой Windows на экран каждый раз, когда пользователь должен уточнить выбранное им действие.

Диалоговое окно содержит набор типовых объектов (элементов) управления, среди которых наиболее часто встречаются:

вкладки, имеющие вид типового диалогового окна, но они расположены в главном диалоговом окне одна под другой, так что видны только их ярлычки. Выбрать вкладку можно щелчком мыши;

командные кнопки, имеющие прямоугольную форму и служащие для выполнения написанных на них команд. Выполнить команду можно щелчком мыши на командной кнопке;

кнопки выбора, имеющие форму круга и предназначенные для выбора одного из нескольких возможных вариантов. Вариант выбирается щелчком мыши на кнопке и отмечается точкой внутри круга;

переключатели (флажки), имеющие квадратную форму и предназначенные для включения или выключения режимов. При щелчке мышью на переключателе в его поле появляется специальный знак или, наоборот, этот знак исчезает,

поля списка, служащие для выбора одного варианта из предлагаемого перечня. Элементы списка прокручиваются в окне при выполнении щелчка на стрелке списка;

текстовые поля, в которые вводятся текст или числовые данные. Для числовых полей значение можно устанавливать с помощью пары кнопок со стрелками, расположенных рядом;

окно предварительного просмотра, в котором отображается объект-документ.

Компьютерная графика - это область информатики, в которой рассматриваются алгоритмы и технологии визуализации данных. Развитие компьютерной графики определяется в основном двумя факторами: реальными потребностями потенциальных пользователей и возможностями аппаратного и программного обеспечения. Потребности потребителей и возможности техники неуклонно растут, и на сегодняшний день компьютерная графика активно используется в самых различных сферах. Можно выделить следующие области применения компьютерной графики:

В большинстве научных статей и отчетов не обойтись без визуализации данных. Достойная форма представления данных - это хорошо структурированная таблица с точными значениями функции в зависимости от некоторых переменных. Но часто более наглядной и эффективной формой визуализации данных является графическая, а, например, при моделировании и обработке изображений - единственно возможная. Некоторые виды отображения информации различного происхождения перечислены в следующей таблице:

Многие программы для финансовых, научных, технических расчётов используют эти и некоторые другие способы визуализации данных. Визуальное представление информации является прекрасным инструментом при проведении научных исследований, наглядным и веским аргументом в научных статьях и дискуссиях.

2.13. Уровни абстракции в процессе разработки программного обеспечения: архитектура, структура, реализация). Цикл разработки прикладного программного обеспечения: концептуализация, анализ, проектирование, кодирование, тестирование, эволюция, сопровождение.

При изучении вопроса необходимо обратить внимание на следующие особенности

Формализованный подход к разработке прикладных программ

Если целевая функция контроллера сформулирована, т.е. задача на разработку поставлена, то для получения текста исходной программы необходимо выполнить ряд последовательных действий:

1. подробное описание задачи;
2. анализ задачи;
3. инженерную интерпретацию задачи, желательно с привлечением того или иного аппарата формализации (граф автомата, сети Петри, матрицы состояний и связности и т.п.);
4. разработку общей блок-схемы алгоритма (БСА) работы конт-роллера;
5. разработку детализированных БСА отдельных процедур, выделенных на основе модульного принципа составления программ;

6. детальную проработку интерфейса контроллера и внесение исправлений в общую и детализированные БСА;

7. распределение рабочих регистров и памяти МК;

8. формирование текста исходной программы.

В результате работы по трем первым пунктам данного перечня получают так называемую функциональную спецификацию прикладной программы МК, в которой основное внимание уделяется детализации способов формирования входной и выходной информации.

На языке схем алгоритмов разработчик описывает метод, выбранный для решения поставленной задачи. Довольно часто бывает, что одна и та же задача может быть решена различными методами. Способ решения задачи, выбранный на этапе ее инженерной интерпретации, на основе которого формируется БСА, определяет не только качество разрабатываемой прикладной программы, но и качественные показатели конечного изделия.

Разработка БСА очень похожа на разработку аппаратурных средств, систем автоматики и обработки данных. В основу разработки БСА положена та же самая процедура модульного проектирования, которая традиционно используется разработчиками аппаратурных средств. Отличие состоит в том, что при разработке аппаратурных средств в качестве ’’строительного” материала используются логические схемы, триггеры, регистры и другие интегральные элементы, а при создании программного обеспечения разработчик оперирует командами, подпрограммами, таблицами и другими программными объектами из арсенала средств обработки данных.

Так как алгоритм есть точно определенная процедура, предписывающая контроллеру однозначно определенные действия по преобразованию ’’сырых” исходных данных в обработанные выходные данные, то разработка БСА требует предельной точности и однозначности используемой атрибутики: символических имен переменных, констант (уставок), подпрограмм (модулей), символических адресов таблиц, портов ввода/ вывода и т.п. Основное внимание при разработке БСА следует уделить тому разделу функциональной спецификации прикладной программы, в котором приводится описание аппаратуры сопряжения МК с объектом управления.

Разработка прикладной программы МК заключается в использовании метода декомпозиции, при котором вся задача последовательно разделяется на меньшие функциональные модули, каждый из которых можно анализировать, разрабатывать и отлаживать отдельно от других. При выполнении прикладной программы в МК управление без всяких двусмысленностей передается от одного функционального модуля к другому. Схема связности этих функциональных модулей, каждый из которых реализует некоторую процедуру, образует общую (или системную) БСА прикладной программы. Это разделение задачи на модули и субмодули выполняется последовательно до такого уровня, когда разработка БСА модуля становится простым и понятным делом. Метод последовательной декомпозиции обладает достаточной гибкостью, что позволяет привести степень детализации БСА в соответствие со сложностью процедуры.

Структурное программирование есть процесс построения прикладной программы из строгого набора программных модулей, каждый из которых реализует определенную процедуру обработки данных. Программные модули должны иметь только одну точку входа и одну точку выхода. Только в этом случае отдельные модули можно разрабатывать и отлаживать независимо, а затем объединять в законченную прикладную программу с минимальными проблемами их взаимосвязей. Источником подавляющего большинства ошибок программирования является использование модулей, имеющих один вход и несколько выходов. При необходимости организации множественных ветвлений в программе декомпозицию задачи выполняют таким образом, чтобы каждый функциональный модуль имел только один вход и один выход.

Вне зависимости от функционального назначения процедуры при разработке ее БСА необходимо придерживаться следующей очередности работы:

1. Определить, что должен делать модуль (это уже было сделано при разработке системной БСА, но теперь разработчик имеет дело с фрагмен-том прикладной программы, а не с целой программой, и, следовательно, может потребоваться доопределение и уточнение целевого назначения процедуры).

2. Определить способы получения модулем исходных данных (от дат-чиков через порты ввода, или из таблиц в памяти, или через рабочие регистры). Для реализации ввода исходных данных в модуль в его БСА надо включить соответствующие операторы.

3. Определить необходимость какой-либо предварительной обработки введенных исходных данных (маскирование, сдвиг, масштабирование, перекодировка). Если до использования ’’сырых” данных требуется их предобработка, то в состав БСА включаются соответствующие операторы.

4. Определить способ преобразования входных данных в требуемые выходные. Используя операторы процедур и условные операторы приня-тия решения, отобразить на языке БСА выбранный метод содержатель-ной обработки исходных данных.

5. Определить способы выдачи из модуля обработанных данных (пе-редать в память, или в вызывавшую программу, или в порты вывода информации) . Необходимые действия отобразить в БСА.

6. Определить необходимость какой-либо постобработки выводимых данных (изменение формата, перекодирование, масштабирование, маскирование) . Ввести в БСА операторы подготовки данных для вывода из модуля.

7. Вернуться к п. 1 настоящего перечня работ и проанализировать по-лученный результат. Выполнить итеративную корректировку БСА с целью сделать ее простой, логичной, стройной и обладающей четким графическим образом.

8. Проверить работоспособность алгоритма на бумаге путем подстановки в него действительных данных. Убедиться в его сходимости и результативности.

9. Рассмотреть предельные случаи и попытаться определить граничные значения информационных объектов, с которыми оперирует алгоритм, за пределами которых он теряет свойства конечности, сходимости или результативности. (Особое внимание при этом следует уделить анализу возможных ситуаций переполнения разрядной сетки МК, изменения знака результата операции, деления на переменную, которая может принять нулевое значение.)

10. Провести мысленный эксперимент по определению работоспособности алгоритма в реальном масштабе времени, когда стохастические события, происходящие в объекте управления, могут оказать влияние на работу алгоритма. При этом самому тщательному анализу следует подвергнуть реакцию алгоритма на возможные прерывания с целью определения критических операторов, которые необходимо защитить от прерываний..

2. Подпрограмма (англ. subroutine) – поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы.

Назначение подпрограмм. Подпрограммы изначально появились как средство оптимизации программ по объёму занимаемой памяти - они позволили не повторять в программе идентичные блоки кода, а описывать их однократно и вызывать по мере необходимости. К настоящему времени данная функция подпрограмм стала вспомогательной, главное их назначение - структуризация программы с целью удобства её понимания и сопровождения.

Выделение набора действий в подпрограмму и вызов её по мере необходимости позволяет логически выделить целостную подзадачу, имеющую типовое решение. Такое действие имеет ещё одно (помимо экономии памяти) преимущество перед повторением однотипных действий: любое изменение (исправление ошибки, оптимизация, расширение функциональности), сделанное в подпрограмме, автоматически отражается на всех её вы-

зовах, в то время как при дублировании каждое изменение необходимо вносить в каждое вхождение изменяемого кода.

Даже в тех случаях, когда в подпрограмму выделяется однократно производимый набор действий, это оправдано, так как позволяет сократить размеры целостных блоков кода, составляющих программу, то есть сделать программу более понятной и обозримой.

Вызов подпрограммы выполняется с помощью команды вызова, включающей в себя имя подпрограммы.

Очень часто требуется из одной подпрограммы обращаться к другой подпрограмме. Такое обращение к подпрограмме называется вложенным. Количество вложенных подпрограмм называется уровнем вложенности подпрограмм. Максимально допустимый уровень вложенности подпрограмм определяется количеством ячеек памяти, предназначенных для хранения адресов возврата из подпрограмм, т.е. размером сегмента стека.

Параметры подпрограмм

Подпрограммы часто используются для многократного выполнения стереотипных действий над различными данными. Подпрограмма обычно имеет доступ к объектам данных, описанным в основной программе, поэтому для того, чтобы передать в подпрограмму обрабатываемые данные, их достаточно присвоить, например, глобальным переменным.

3. Цикл – разновидность управляющей конструкции, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая многократно исполняемая последовательность инструкций, организованная любым способом (например, с помощью условного перехода).

Последовательность инструкций, предназначенная для многократного исполнения, называется телом цикла. Единичное выполнение тела цикла называется итерацией. Выражение определяющее, будет в очередной раз выполняться итерация, или цикл завершится, называется условием выхода или условием окончания цикла (либо условием продолжения в зависимости от того, как интерпретируется его истинность - как признак необходимости завершения или продолжения цикла). Переменная, хранящая текущий номер итерации, называется счётчиком итераций цикла или просто счётчиком цикла. Цикл не обязательно содержит счётчик, счётчик не обязан быть один - условие выхода из цикла может зависеть от нескольких изменяемых в цикле переменных, а может определяться внешними условиями (например, наступлением определённого времени), в последнем случае счётчик может вообще не понадобиться.

Исполнение любого цикла включает первоначальную инициализацию переменных цикла, проверку условия выхода, исполнение тела цикла и обновление переменной цикла на каждой итерации.

Виды циклов.

Иногда в программах используются циклы, выход из которых не предусмотрен логикой программы. Такие циклы называются безусловными, или бесконечными. Специальных синтаксических средств для создания бесконечных циклов, ввиду их нетипичности, языки программирования не предусматривают, поэтому такие циклы создаются с помощью конструкций, предназначенных для создания обычных (или условных) циклов – в ассемблере – с помощью команд безусловных переходов.

Цикл с предусловием – цикл, который выполняется пока истинно условие, указанное перед его началом. Это условие проверяется до выполнения тела цикла, поэтому тело может быть не выполнено ни разу (если условие с самого начала ложно).

Цикл с постусловием – цикл, в котором условие проверяется после выполнения тела цикла. Отсюда следует, что тело всегда выполняется хотя бы один раз.

Цикл с выходом из середины Синтаксически такой цикл оформляется с помощью трёх конструкций: начала цикла, конца цикла и команды выхода из цикла. Конструкция начала маркирует точку программы, в которой начинается тело цикла, конструкция конца – точку, где тело заканчивается. Внутри тела должна присутствовать команда выхода из

цикла, при выполнении которой цикл заканчивается и управление передаётся на оператор, следующий за конструкцией конца цикла.

Цикл со счётчиком – цикл, в котором некоторая переменная изменяет своё значение от заданного начального значения до конечного значения с некоторым шагом, и для каждого значения этой переменной тело цикла выполняется один раз.

4. Группа команд передачи управления

Группа представлена командами безусловного и условного переходов, командами вызова подпрограмм и командами возврата из подпрограмм.

2.14. Критерии оценки качества программы. Средства и инструменты разработки программного обеспечения. Стиль программирования. Организация разработки программного обеспечения.

При изучении вопроса необходимо обратить внимание на следующие особенности.

Критерии оценки качества программы

Можно привести множество различных критериев для оценки качества программного продукта. Обозначим 4 наиболее важных и характерных критерия:

1. надежность
2. использование ОЗУ
3. время выполнения
4. дружественность пользовательского интерфейса

Средства и инструменты разработки программного обеспечения.

При разработке программного продукта задействуется довольно большой спектр инструментального ПО, которое решает некоторые специальные задачи. Довольно условно их можно разбить на четыре группы:

a) **необходимые** – те, без которых невозможно в принципе получить исполняемый код;

К необходимым можно отнести:

- редакторы текстов;
- компиляторы и ассемблеры;
- компоновщики или редакторы связей (linkers);

b) **часто используемые** – средства, использования которых, в отличие от необходимых, можно избежать. Но без них процесс разработки весьма затрудняется и удлиняется;

Из часто используемых средств стоит назвать:

- утилиты автоматической сборки проекта;
- отладчики;
- программы создания инсталляторов;
- редакторы ресурсов;
- профилировщики;
- программы поддержки версий;
- программы создания файлов помощи (документации).

c) **специализированные** – используются в исключительных случаях, решают довольно специфичные задачи:

- программы отслеживания зависимостей;
- дизассемблеры;
- декомпиляторы;
- hex-редакторы;

- программы отслеживания активности системы и изменений, происходящих в системе;
- программы-вериферы и контейнеры (создают виртуальную среду для отдельных классов программ, в которой можно исследовать поведение программы)
- и т.д.

d) **интегрированные среды** – содержат большую часть из приведенных выше программ и позволяют осуществлять

В каждом классе существуют огромное число продуктов, каждый со своими особенностями, достоинствами и недостатками.

Дадим краткую характеристику названным классам программ и приведем некоторые критерии оценки, по которым можно сравнивать программы из одного класса.

Но сначала укажем на характеристики, универсальные для всех программ:

- фирма-производитель, автор (зачастую имя производителя значит больше, чем все остальное).
- название продукта;
- номер последней версии;
- класс продукта, который установил для него производитель (например, Hackers-Viewer, который включает в себя неплохой дизассемблер и редактор PE-файлов, поставляется просто как hex-редактор);
- тип дистрибьюции программы (с открытыми кодами/бесплатная (freeware)/условно-бесплатная (shareware)/платная) и стоимость;
- наличие и тип поддержки, ее стоимость;
- доступность и качество документации;
- простота и понятность интерфейса;
- наличие пробных версий (для платных программ);
- сайт программы и возможность ее скачки;
- размер дистрибутива и его состав;
- дополнительные (не основные) возможности, предоставляемые программой;

1.Обычные программы (не интегрированные среды)

1.1. Компиляторы (ассемблеры) и редакторы связей

Эти два класса программ следует объединить, т.к. в поставку любого современного компилятора входит и редактор связей.

Компилятор (ассемблер) формирует объектный код, переводя программу с языка программирования (языка ассемблера), а редактор формирует исполнимый файл, собирая объектные и библиотечные файлы и редактируя перекрестные ссылки.

Для компиляторов можно указать следующие характеристики:

- язык, с которого производится компиляция;
- диалект/стандарт языка;
- аппаратные платформы и ОС, для которых может формироваться объектный и исполнимый файл;
- наличие возможности и качество оптимизации кода;
- форматы поддерживаемых объектных, библиотечных и исполнимых файлов;

1.2. Редакторы текстов

Предназначены для ввода и корректировки текстов программ. Могут быть как общими, так и предназначенными для поддержки конкретного языка(ов) и/или сред(ы).

Характеристики:

- формат и кодировка обрабатываемых файлов;
- возможность выделения лексем в тексте;
- возможность поддержки оформления текста в соответствии с парадигмами языка;
- возможность вызывать процесс компиляции прямо из редактора;
- возможность генерации части текста программы (чаще бывает не у редакторов, а у сред).

1.3. Отладчики

Предназначены для пошагового отслеживания работы программы, слежения за изменением ее и системных переменных, изменением состояния процессора во время работы программы и т.д.

Различают два основных типа отладчиков:

- отладчики пользовательского режима;
- отладчики режима ядра;

Первые могут лишь следить за работой программ пользовательского режима и не способны ни отслеживать системные вызовы, ни следить за работой ядра. Кроме того, для использования таких отладчиков программа должна быть соответствующим образом подготовлена (скомпилирована).

Отладчики же режима ядра, напротив, позволяют полностью контролировать работу системы, а, следовательно, и всех программ.

Характеристики:

- тип (режима ядра/пользовательский);
- поддержка символьной отладки (способность читать исходные коды программы и работать с ними). Набор поддерживаемых языков (сред/диалектов);
- набор отображаемой информации: регистры процессора, стек, память (режимы отображения содержимого памяти);
- поддерживаемые режимы отладки: пошаговый, с точками останова, с реакцией на события в системе;
- состав отслеживаемых событий в системе: аппаратные прерывания, обращения к драйверу (другому модулю ядра), вызов функции и т.д.
- (обычно для отладчиков режима ядра) требования к аппаратной поддержке, возможность работы на «живой» системе;
- возможность анализа файлов дампа.

1.4. Программы создания инсталляторов

Предназначены для создания дистрибутивов программ и пакетов программ.

Задачи, выполняемые подобными программами для различных платформ, могут сильно различаться. Мало того, с выходом Windows Installer и опубликования его API для платформы Win32 началось разделение программ на поддерживающие WI и использующие свои средств.

Как правило, все дистрибутивы имеют интерфейс программ-мастеров (т.е. пошаговое уточнение настроек). Кроме того, почти всегда имеется возможность удаления установленной программы.

Характеристики:

- ориентированны на использование Windows Installer или используют свои средства;
- возможность автоматического отслеживания зависимостей исполнимых файлов и разделяемых библиотек;
- наличие встроенного языка сценариев;
- возможность и пределы, в которых можно изменять поведения мастера инсталляции;
- возможность использования и поддержка национальных языков;
- функции, поддерживаемые в процессе установки (кроме копирования файлов):

a) создание ключей реестра;
b) регистрация COM-объектов;
c) перезагрузка системы после или в процессе установки;

- возможность удаления установленной программы;
- возможность контроля версий устанавливаемой программы (перезапись, если необходимо) и разделяемых библиотек;
- возможность и степень сжатия дистрибутива;

- возможность создания дистрибутива, состоящего из одного, или заданного количества файлов;

1.5. Редакторы ресурсов

Создают и обрабатывают файлы ресурсов, которые после обработки могут быть скомпилированы и включены в исполнимый модуль. Эти программы специфичны для платформы Win.

Характеристики:

- состав поддерживаемых ресурсов;
- возможность работы с нестандартными ресурсами;
- возможности импорта и экспорта ресурсов.

1.6. Профилировщики

Дают возможность отслеживать время работы программы в целом и отдельных ее частей.

Характеристики:

- поддерживаемые платформы;
- возможности кросс-профилировки (эмуляции системы);
- вид выдаваемых данных (графики, гистограммы, таблицы);

1.7. Программы поддержки версий

1.8. Программы создания файлов помощи (документации).

Позволяют создавать файлы помощи, автоматизировать документирование.

Характеристики:

- форматы поддерживаемых выходных файлов (hlp, chm, html, pdf, …);
- средства, необходимые для работы с файлами документации;
- возможность конвертирования из других распространенных форматов;
- возможность структурирования информации в файле помощи;
- возможность организации поиска по документации;
- возможность интеграции в существующие среды (например, для библиотеки COM – объектов возможность вызывать справку по ней при нажатии F1, если курсор стоит на объекте из этой библиотеки)
- возможность автоматической генерации помощи (или заготовки) по описанию библиотеки, COM-объекта.

1.9. Дизассемблеры и декомпиляторы;

Предназначены для получения исходного кода на языке программирования из исполняемого модуля.

Характеристики:

- поддерживаемые языки (компиляторы).
- возможность использования символьной информации о файле (отладочной и др.)
- возможность интерактивной работы с листингом (замены имен переменных и функций, отслеживания вызовов, модификация кода)

1.10. Программы отслеживания активности системы и изменений, происходящих в системе;

Позволяют отслеживать действия программ по изменению реестра, файловой системы, вызовов системных сервисов и т.д. Следят за загруженностью системы в целом.

Характеристики

- тип отслеживаемых изменений/активности;
- возможность протоколирования (логирования);
- возможность фильтрации получаемой информации;
- возможность уведомления;

1.11. Программы-вериферы и контейнеры.

Создают виртуальную среду для отдельных классов программ, в которой можно исследовать поведение программы):

1.12. Интегрированные среды

Включают в себя большую часть выше перечисленных средств и обеспечивают их взаимосвязь.

Стиль программирования - набор приемов или методов программирования, которые используют программисты, чтобы получить правильные, эффективные, удобные для применения и легкочитаемые программы. (Словарь технических терминов)

Стиль программирования, под которым понимают стиль оформления программ и их «структурность», также существенно влияет на читаемость программного кода и количество ошибок программирования.

Основные принципы форматирования

Основная теорема форматирования гласит, что хорошее визуальное форматирование показывает логическую структуру программы. Создание красивого кода - хорошо, а демонстрация структуры кода - лучше. Если одна методика лучше показывает структуру кода, а другая выглядит красивей, следует использовать ту, которая лучше демонстрирует структуру.

Форматирование - это ключ к структуре программы. Компьютеру важна исключительно информация о скобках или операторах begin и end, а человек склонен делать выводы из визуального представления кода.

Цели хорошего форматирования

Хорошая схема форматирования должна делать следующее:

точно представлять логическую структуру кода. Для демонстрации логической структуры программисты обычно применяют отступы и другие неотображаемые символы;

единообразно показывать логическую структуру кода. Некоторые стили форматирования состоят из правил с таким количеством исключений, что последовательно их соблюдать практически невозможно. Действительно хороший стиль подходит в большинстве случаев;

улучшать читабельность. Стратегия использования отступов, соответствующая логике, но усложняющая процесс чтения кода, бесполезна. Схема форматирования, использующая пробелы и разделители только там, где они требуются компилятору, логична, но читать такой код невозможно. Хорошая структура форматирования упрощает чтение кода;

выдерживать процедуру исправления. Лучшие схемы форматирования хорошо переносят модификацию кода. Исправление одной строки не должно приводить к изменению нескольких других.

В дополнение к этим критериям иногда во внимание принимается и задача минимизации количества строк кода, необходимых для реализации простого выражения или блока.

Способы форматирования

Можно получить хороший формат кода, по-разному используя несколько инструментов для форматирования.

• Неотображаемые символы

Используйте неотображаемые символы для улучшения читаемости. Неотображаемые символы, к которым относятся пробелы, знаки табуляции, переводы строк и пустые строки, - это основное средство для демонстрации структуры программы.

Группировка взаимосвязанных выражений - еще один способ применения неотображаемых символов.

В литературе мысли группируются в абзацы. Хорошо написанный абзац содержит предложения, относящиеся только к определенной идее. Он не должен содержать посторонних предложений. Точно так же абзац кода должен содержать только взаимосвязанные операторы, выполняющие одно задание.

Пустые строки. Кроме необходимости группировать взаимосвязанные операторы, очень важно отделять несвязанные выражения друг от друга. Начало нового абзаца в книге обозначается отступом или пустой строкой. Начало нового абзаца в коде нужно указывать с помощью пустой строки.

Пустые строки позволяют продемонстрировать организацию программы. Вы можете использовать их для деления групп взаимосвязанных операторов на абзацы, отделения методов друг от друга и выделения комментариев.

Отступы. Применяйте отступы для демонстрации логической структуры программы. Как правило, операторы выделяются отступами, когда они следуют после некоторого выражения, от которого они логически зависят. Оптимальными являются отступы из 2-4 пробелов.

Скобки Используйте скобки чаще, чем вам это кажется необходимым. Применяйте скобки для разъяснения выражений, состоящих из двух и более членов. Возможно, в скобках нет нужды, но они добавляют ясности и ничего вам не стоят.

стили программирования

- Программирование от состояний;
- Структурное программирование;
- Сентенциальное программирование;
- Программирование от событий;
- Программирование от процессов и приоритетов;
- В программировании от состояний процесс представляется как смена состояний системы. Новое состояние возникает в результате действия, изменяющего старое состояние, а выбор этого действия зависит от проверки условий. Математической моделью программы служит конечный автомат. Инвариантом применимости данного стиля является следующая характеристика задачи:
- Действия глобальны, условия локальны.
- Естественным способом программирования такой задачи на современных языках программирования является использование операторов goto либо объектов, обменивающихся информацией через общее поле памяти.
- В структурном программировании, которому сейчас учат как монопольному первоуровневому стилю, действия и условия локальны. Управляющие действия образуют иерархическую структуру, а потоки передачи данных в принципе должны согласовываться с данной структурой. Этот стиль поддерживается структурами современных традиционных языков программирования (например, Pascal, C). Математической моделью программ являются здесь вычислимые функции.
- В сентенциальном стиле (Рефал, Пролог) действия и условия глобальны. Каждый шаг программы проверяет все поле зрения на соответствие образцу, находит тем самым применимое правило преобразования, и, согласно найденному правилу, преобразует все поле памяти.
- Программирование на Прологе традиционно называют логическим. От логики здесь ничего не осталось, и термин вводит в заблуждение
- В программировании от событий и от приоритетов действия локальны, условия глобальны. Условие состоит в том, что в системе произошло некоторое событие, лучшим обработчиком которого оказалось данное действие. Но в программировании от событий такое событие снабжает процесс-обработчик важной информацией. А при программировании от приоритетов событие состоит в том, что все более приоритетные процессы ничего не могут сделать, и никакой позитивной информации активизируемому процессу не дает.
- На уровне понятий более высокого уровня возникают другие стили.
- Функциональный стиль программирования, когда программа представляет собой функционал высокого уровня, преобразующий функции, представлен языками ЛИСП и ML. Если функции типизированы, то этот подход, сохраняя возможности понятий высших уровней исключительно компактно выразить сложную структуру, вдобавок крайне эффективен по использованию ресурсов. Но нынешние реализации функционального программирования не могут удержаться от использования рекурсий нетипизируемых конструкций типа оператора вычисления произвольного выражения или оператора неподвижной точки. Это создало функциональному программированию репутацию крайне неэф-

фективного стиля, подходящего лишь для прототипов программ. Функциональному стилю соответствуют интуиционистская логика предикатов (типизированный вариант) и комбинаторная логика (нетипизированный вариант).

• Объектно-ориентированый стиль является четверть-шагом к функциональному с точки зрения теории. Для сложных систем работа с действиями эффективнее работы с данными.

Организация разработки программного обеспечения.

Одним из главных критериев при выборе компании-разработчика является зрелость организации. Незрелой считается компания, где организация разработки программного обеспечения зависит только от конкретных исполнителей и менеджеров, а решения чаще всего генерируются на скорую руку. В таком случае есть вероятность превышения бюджета, а также сроков разработки проекта, вот почему менеджерам и разработчикам подобных компаний приходится заниматься исключительно решением актуальных проблем, становясь, таким образом, заложниками собственного программного продукта.

В зрелых же компаниях организация разработки программного обеспечения четко расписана, а механизмы управления проектами отработаны. Все механизмы и процедуры по мере необходимости совершенствуются в пробных проектах, а оценки затрат времени и стоимости проекта основываются на накопленном опыте и являются достаточно точными. Кроме того, в таких компаниях есть определенные стандарты на процессы разработки, тестирования, внедрения, правила оформления конечного программного кода, интерфейса, компонентов и так далее. Все это вместе составляет инфраструктуру и корпоративную культуру, которая поддерживает процесс разработки ПО. Технология выпуска может незначительно меняться от проекта к проекту на основе абсолютно стабильных и проверенных подходов.

В наших же реальных условиях организация разработки программного обеспечения не является огромной. Всё зависит от объёма работы и величины компании, разрабатывающей программное обеспечение. Хороший процесс можно организовать для любого проекта - как для гиганта, требующего для его осуществления двести разработчиков, так и для небольшого проекта на несколько разработчиков.

Цель организации разработки программного обеспечения состоит не в том, чтобы усложнить работу разработчикам и "убить" творческое начало проекта необходимостью писать большое количество документации. Такой процесс нужен затем, чтобы компания веб услуг, разрабатывающая программное обеспечение, могла предсказуемо, с соблюдением бюджета проекта и сроков его исполнения создать программное обеспечение высокого качества, которое бы удовлетворяло все требования пользователей.