Учреждение образования
«Белорусский государственный университет культуры и искусств»
Факультет культурологии и социально-культурной деятельности
Кафедра информационных технологий в культуре

СОГЛАСОВАНО
Заведующий кафедрой
_____ Т.С. Жилинская
«28» 10 2024 г.

СОГЛАСОВАНО
Декан факультета
_____ Н.Е. Шелупенко
«28» 10 2024 г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ **ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ**

для специальности
6-05-0314-03 «Социально-культурный менеджмент и коммуникации»,
профилизации «Мультимедийные технологии и цифровые коммуникации»

СОСТАВИТЕЛЬ:
Т. И. Песецкая, доцент кафедры информационных технологий в культуре учреждения образования «Белорусский государственный университет культуры и искусств», кандидат физико-математических наук

Советом факультета культурологии и социокультурной деятельности учреждения образования «Белорусский государственный университет культуры и искусств» (протокол № 10 от 27.06.2025)

СОСТАВИТЕЛЬ:

Т. И. Песецкая, доцент кафедры информационных технологий в культуре учреждения образования «Белорусский государственный университет культуры и искусств», кандидат физико-математических наук

Рецензенты:

кафедра дискретной математики и алгоритмики ФПМИ, Белорусского государственного университета, заведующий кафедрой В.М. Котов, доктор физ.-мат. наук, профессор;

Д.В. Морозов, директор государственного учреждения «Национальное агентство по туризму», кандидат исторических наук

Рассмотрен и рекомендован к утверждению:

кафедрой информационных технологий в культуре учреждения образования «Белорусский государственный университет культуры и искусств» *(протокол № 9 от 22.05.2025);*

Советом факультета культурологии и социокультурной деятельности (протокол от 27.__06__.2025 г., №10).

ОГЛАВЛЕНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Учебная дисциплина «Визуальное программирование» предназначена для студентов специальности 6-05-0314-03 Социально-культурный менеджмент и коммуникации, профилизации «Мультимедийные технологии и цифровые коммуникации».

Изучение учебной дисциплины «Визуальное программирование» направлено на обучение студентов теоретическим основам и методам анализа данных сферы культуры, и практическим подходам визуального программирования с целью создания информационных ресурсов в сфере культуры.

Целью преподавания учебной дисциплины является приобретение студентами специальности 6-05-0314-03 Социально-культурный менеджмент и коммуникации, будущими специалистами по управлению и коммуникациям, необходимого уровня компетентности в области визуального программирования с целью создания информационных ресурсов в сфере культуры.

Для достижения поставленной цели необходимо решение следующих задач:

1) формирование системы базовых знаний визуального программирования;

2) изучение основных принципов языков визуального программирования;

3) выработка навыков и умений использования визуальных сред разработки программ;

4) изучение возможностей разработки мобильных приложений на базе языков визуального программирования;

5) освоение подходов к использованию визуального программирования для создания динамической 2D и 3D векторной графики;

Знания и навыки, полученные при изучении учебной дисциплины «Визуальное программирование», необходимы при изучении таких учебных дисциплин, как: «Технологии 3D-анимации», «Мультимедийные технологии художественного проектирования», «Моушн дизайн».

В соответствии с учебным планом учреждения высшего образования по специальности 6-05-0314-03 Социально-культурный менеджмент и коммуникации, профилизации «Мультимедийные технологии и цифровые

коммуникации» освоение образовательной программы по учебной дисциплине «Визуальное программирование» должно обеспечивать формирование следующей специальной компетенции:

СК-30. Использование визуального программирования в создании приложений социально-культурной сферы.

В результате изучения учебной дисциплины «Визуальное программирование» студенты должны *знать*:

– роль и месть визуального программирования в системе научных знаний;

– основы языков визуального программирования;

– современные программные среды визуального программирования;

– подходы к использованию языков визуального программирования для создания динамической графики;

– основы разработки мобильных приложений с помощью языков визуального программирования;

Студенты должны *уметь*:

– разрабатывать алгоритмы для решения задач социально-культурной сферы с использованием языков визуального программирования;

– использовать соответствующие среды визуального программирования для решения конкретных задач;

– применять средства визуального программирования для создания динамической графики;

– создавать мобильные приложения с помощью языков визуального программирования.

Студенты должны приобрести *навыки*:

– анализа информационных задач социально-культурной сферы и подбора средств визуального программирования для их решения;

– освоения инструментария различных сред визуального программирования, предназначенных для решения конкретных задач социально-культурной сферы.

– использования средств визуального программирования в приложениях для создания динамической графики;

– разработки мобильных приложений с использованием сред визуального программирования.

Методы и технологии обучения.

На лекциях особое внимание уделяется рассмотрению теоретических аспектов решения задач сферы культуры с использованием языков и сред визуального программирования. Лабораторные занятия направлены на формирование умений практического использования полученных знаний для решения прикладных задач социально-культурной сферы.

Учебным планом на изучение учебной дисциплины «Визуальное программирование» всего предусмотрено 90 часов, из них 34 часа – аудиторные занятия. Примерное распределение аудиторных часов по видам занятий: лекции – 6 часов, практические занятия – 28 часов.

Дисциплина рассчитана на один семестр. Текущий контроль осуществляется при выполнении и сдаче отчетов лабораторных работ. Промежуточная форма контроля – устный опрос. Рекомендуемая форма контроля знаний – зачёт.

ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

Тема 1. Введение в визуальное программирование

Визуальное программирование (ВП) представляет собой подход к разработке программ, основанный на использовании графических элементов вместо традиционного текстового кода. Вместо написания строк кода, разработчики используют визуальные блоки, диаграммы и другие графические инструменты для построения программной логики. ВП быстро набирает популярность, предлагая ряд преимуществ как для начинающих, так и для опытных программистов.

Основные концепции ВП:

– ВП использует визуальные элементы (графические элементы), такие как блоки, иконки, линии и диаграммы, для представления программного кода и логики.

– ВП нацелено на интуитивность, что делает процесс программирования более понятным, особенно для пользователей без опыта программирования.

– Основным принципом ВП является упрощение процессов разработки программ, с целью доступности программирования для широкого круга пользователей.

Преимущества ВП:

– Доступность: ВП снижает порог входа в мир программирования, делая его доступным для детей, художников, дизайнеров и других пользователей без технического опыта.

– Скорость разработки: Визуальные инструменты позволяют быстро создавать прототипы, что ускоряет процесс разработки.

– Улучшенная отладка: Визуальное представление кода облегчает поиск и исправление ошибок.

– Простая коммуникация: Визуальные диаграммы облегчают понимание логики программы как разработчикам, так и заказчикам.

Типы ВП:

– Блочное программирование использует графические блоки, представляющие отдельные операции или функции. Блоки соединяются друг с другом для создания программной логики.

– Диаграммы потоков данных визуализируют поток данных в программе.

– Графические языки программирования используют визуальные элементы для представления кода.

– Визуальные редакторы кода представляют текстовый код в визуальном формате, делая его более понятным.

ВП делает программирование более доступным, интуитивным и эффективным, открывая новые возможности для широкого круга пользователей. С развитием технологий ВП будет играть все более важную роль в различных сферах жизни, от образования до разработки сложных систем.

Графическое представление потока данных является неотъемлемой частью визуального программирования, повышая понятность, эффективность и гибкость разработки программ.

Диаграмма потока данных (Data Flow Diagram / DDF) – это графическое представление потока данных через систему. Она используется для визуализации процесса обработки данных и помогает понять, как данные перемещаются между различными компонентами системы. DFD является важным инструментом в системном анализе и проектировании, особенно для новичков, так как она предоставляет наглядное представление о работе системы. Визуализация данных с помощью DFD позволяет выявить узкие места, избыточные процессы и потенциальные улучшения в системе. Это особенно полезно на начальных этапах проектирования, когда важно получить общее представление о системе.

Основными компонентами DFD являются:

Процессы

Процессы обозначаются кругами или овалами и представляют собой действия или функции, которые преобразуют входные данные в выходные. Например, процесс "Обработка заказа" может принимать данные о заказе и преобразовывать их в данные о подтверждении заказа. Процессы могут быть простыми, такими как "Ввод данных", или сложными, такими как "Анализ данных". Важно, чтобы каждый процесс был четко определен и имел понятное название, отражающее его функцию в системе. Например, процесс "Регистрация пользователя" может включать в себя несколько подзадач, таких как проверка данных, создание учетной записи и отправка подтверждения.

Потоки данных

Потоки данных изображаются стрелками и показывают направление движения данных между процессами, хранилищами данных и внешними сущностями. Например, поток данных "Заказ" может идти от клиента к процессу "Обработка заказа". Потоки данных могут быть однонаправленными или двунаправленными, в зависимости от того, как данные перемещаются между компонентами системы. Например, поток данных "Запрос информации" может идти от пользователя к процессу "Поиск информации", а поток данных "Ответ на запрос" — обратно к пользователю. Важно, чтобы каждый поток данных был четко обозначен и имел понятное название, отражающее его содержание.

Хранилища данных

Хранилища данных изображаются прямоугольниками с открытым правым краем и представляют собой места, где данные хранятся. Например, хранилище данных "База данных клиентов" может содержать информацию о клиентах. Хранилища данных могут быть физическими, такими как базы данных и файлы, или логическими, такими как временные хранилища данных в памяти системы. Важно, чтобы каждое хранилище данных было четко обозначено и имело понятное название, отражающее его содержание. Например, хранилище данных "Лог событий" может содержать информацию о всех событиях, произошедших в системе.

Внешние сущности

Внешние сущности изображаются прямоугольниками и представляют собой объекты или лица, которые взаимодействуют с системой, но не являются её частью. Например, внешняя сущность "Клиент" может отправлять заказы в систему. Внешние сущности могут быть пользователями, другими системами или любыми другими объектами, которые взаимодействуют с системой. Важно, чтобы каждая внешняя сущность была четко обозначена и имела понятное название, отражающее её роль в системе. Например, внешняя сущность "Поставщик" может отправлять данные о поставках в систему.

Этапы Создания Диаграммы потока данных

1. *Определение границ системы*

Первым шагом является определение границ системы, то есть какие процессы, данные и внешние сущности будут включены в диаграмму. Это поможет сфокусироваться на ключевых аспектах системы. Определение

границ системы включает в себя идентификацию всех входов и выходов системы, а также всех взаимодействий с внешними сущностями. Например, для системы обработки заказов границы могут включать процессы от получения заказа до его выполнения и отправки подтверждения клиенту.

2. *Идентификация процессов*

Определите основные процессы, которые происходят в системе. Каждый процесс должен быть описан кратко и ясно, чтобы было понятно, какие данные он принимает и какие данные он выдает. Идентификация процессов включает в себя анализ всех функций и операций, выполняемых системой. Например, для системы управления библиотекой процессы могут включать "Поиск книги", "Выдача книги" и "Возврат книги". Важно, чтобы каждый процесс был четко определен и имел понятное название, отражающее его функцию в системе.

3. *Определение потоков данных*

Идентифицируйте потоки данных между процессами, хранилищами данных и внешними сущностями. Убедитесь, что каждый поток данных имеет четкое направление и название. Определение потоков данных включает в себя анализ всех взаимодействий между компонентами системы. Например, для системы обработки заказов потоки данных могут включать "Заказ", "Подтверждение заказа" и "Информация о заказе". Важно, чтобы каждый поток данных был четко обозначен и имел понятное название, отражающее его содержание.

4. *Определение хранилищ данных*

Определите, где данные будут храниться в системе. Это могут быть базы данных, файлы или другие хранилища данных. Определение хранилищ данных включает в себя анализ всех мест, где данные хранятся в системе. Например, для системы управления библиотекой хранилища данных могут включать "Каталог книг", "Информация о пользователях" и "История выдач". Важно, чтобы каждое хранилище данных было четко обозначено и имело понятное название, отражающее его содержание.

5. *Создание диаграммы*

Используя информацию, собранную на предыдущих шагах, создайте диаграмму потока данных. Убедитесь, что все компоненты правильно связаны и отображают реальный поток данных в системе. Создание диаграммы включает в себя визуализацию всех процессов, потоков данных, хранилищ данных и внешних сущностей. Например, для системы обработки

заказов диаграмма может включать процессы "Получение заказа", "Обработка заказа" и "Отправка подтверждения", а также потоки данных "Заказ", "Подтверждение заказа" и "Информация о заказе". Важно, чтобы диаграмма была четкой и понятной, и чтобы все компоненты были правильно связаны.

Преимущества графического представления потока данных в ВП:

Повышенная наглядность. Визуализация потока данных позволяет легко увидеть, как информация передается между компонентами системы, что делает программу более понятной как для разработчика, так и для заказчика.

Упрощение анализа. Графическое представление упрощает анализ структуры системы, помогая обнаружить узкие места, дублирование кода и другие проблемы.

Улучшение коммуникации. Визуальные диаграммы облегчают обсуждение и документирование структуры системы между разработчиками, заказчиками и другими акторами.

Повышение гибкости. Графическое представление позволяет легко изменять структуру системы и добавлять новые компоненты.

Диаграммы UML. Язык моделирования UML (Unified Modeling Language) предоставляет широкий спектр диаграмм, включая диаграммы классов, диаграммы последовательности и диаграммы компонентов, которые могут быть использованы для визуализации потока данных в системе.

Блочные диаграммы. В блочных диаграммах каждый компонент системы представлен блоком, а стрелки между блоками отражают поток данных. Блочные диаграммы часто используются в визуальных языках программирования, таких как ***Scratch и Blockly.***

Инструменты для графического представления потока данных: Visio – Популярный программный пакет для создания различных визуальных диаграмм, включая DFD и UML диаграммы; Lucidchart – онлайн инструмент для создания диаграмм, который предлагает широкий спектр шаблонов и функций для визуализации потока данных; Draw.io – бесплатный инструмент для создания диаграмм, доступный в виде веб-приложения и расширения для Google Docs и Confluence.

Проектирование визуальных алгоритмов

При проектировании визуальных алгоритмов используют специальные графические элементы, называемые графически блоками. Результатом алгоритмизации решения задачи является блок-схема алгоритма, состоящая

из некоторой последовательности таких графических блоков.

Рисунок 1 – Блок-схема алгоритма

Общими правилами при проектировании визуальных алгоритмов являются следующие:

В начале алгоритма должны быть блоки ввода значений входных данных.

После ввода значений входных данных могут следовать блоки обработки и блоки условия.

В конце алгоритма должны располагаться блоки вывода значений выходных данных.

В алгоритме должен быть только один блок начала и один блок окончания.

Связи между блоками указываются направленными или ненаправленными линиями.

Этап проектирования алгоритма следует за этапом формального решения задачи, на котором определены входные и выходные данные, а также зависимости между ними.

При построении алгоритмов для сложной задачи используют системный подход с использованием декомпозиции (нисходящее проектирование сверху-вниз). Как и при разработке любой сложной системы,

при построении алгоритма используют дедуктивный и индуктивный методы. При дедуктивном методе рассматривается частный случай общеизвестных алгоритмов. Индуктивный метод применяют в случае, когда не существует общих алгоритмических решений. Одним из системных методов разработки алгоритмов является метод структурной алгоритмизации. Этот метод основан на визуальном представлении алгоритма в виде последовательности управляющих структурных фрагментов. Выделяют три базовые управляющие процессом обработки информации структуры: композицию, альтернативу и итерацию. С помощью этих структур можно описать любые процессы обработки информации.

Композиция (следование).

Альтернатива.

Итерация.

В соответствии с наличием в алгоритмах управляющих структур композиции, альтернативы и итерации алгоритмы классифицируют на: линейные, разветвленные и циклические алгоритмы.

Линейные алгоритмы не содержат блока условия. Они предназначены для представления линейных процессов. Такие алгоритмы применяют для описания обобщенного решения задачи в виде последовательности модулей.

Разветвленный алгоритм – это алгоритм, который выполняет разные действия в зависимости от условий. Проще говоря, он "ветвится", выбирая один из нескольких возможных путей выполнения в зависимости от выполнения определенного условия.

Циклический алгоритм – это алгоритм, который повторяет набор инструкций до тех пор, пока не будет выполнено определенное условие.

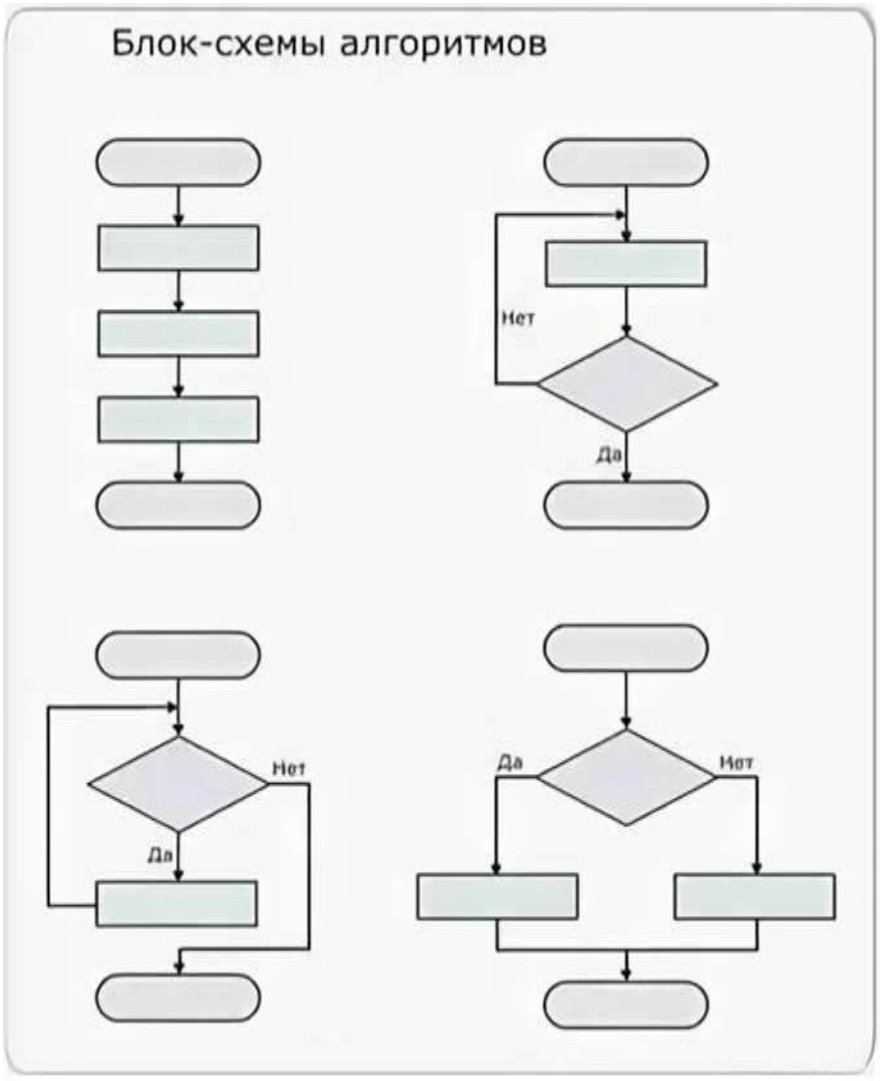


Рисунок 2 – Блок-схемы алгоритмов

Основные элементы разветвленного алгоритма:

Условие – это выражение, которое может быть истинным (true) или ложным (false).

Ветви – это блоки кода, которые выполняются, если условие истинно или ложно.

Типы разветвлений:

Если-иначе (if-else) выполняет одну ветвь, если условие истинно, и другую ветвь, если условие ложно.

Если-иначе-если (if-else-if) выполняет одну из нескольких ветвей в зависимости от выполнения нескольких условий.

Переключатель (switch) выполняет одну ветвь в зависимости от значения переменной.

Основные элементы циклического алгоритма:

Инициализация – начальное значение для счетчика или других переменных.

Условие – выражение, которое проверяется на каждой итерации цикла.

Тело цикла – Блок кода, который выполняется на каждой итерации.

Изменение – Обновление счетчика или других переменных после каждой итерации.

Типы циклов:

Цикл for: Выполняет блок кода определенное количество раз.

Цикл while: Выполняет блок кода, пока условие истинно.

Цикл do-while: Выполняет блок кода хотя бы один раз, а затем повторяет его, пока условие истинно.

Разветвленные и циклические алгоритмы часто используются совместно в программах. Например, цикл может включать в себя проверку условий и ветвление, а ветвление может происходить внутри цикла.

Событийно-ориентированное программирование

Событийно-ориентированное программирование, СОП, создание событийно-управляемых программ, event-driven programming - парадигма программирования, в которой выполнение программы определяется событиями - действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета).

СОП можно также определить, как способ построения компьютерной программы, при котором в коде (как правило, в головной функции программы) явным образом выделяется главный цикл приложения, тело которого состоит из двух частей: выборки события и обработки события.

Как правило, в реальных задачах оказывается недопустимым длительное выполнение обработчика события, поскольку при этом программа не может реагировать на другие события. В связи с этим при написании событийно-ориентированных программ часто применяют автоматное программирование.

Событийно-ориентированное программирование, как правило, применяется в следующих случаях:

– построение пользовательских интерфейсов (в том числе графических);

– создание серверных приложений;

– моделирование сложных систем;

– параллельные вычисления;

– автоматические системы управления, SCADA;

– программирование игр, в которых осуществляется управление множеством объектов.

Визуальное программирование (ВП) выходит за рамки технических задач и открывает новые возможности для творческого самовыражения в сфере культуры. Его интуитивность и доступность делают ВП идеальным инструментом для расширения творческих границ, а именно для создания: мультимедийных проектов, где ВП позволяет создавать интерактивные инсталляции, анимационные фильмы и другие мультимедийные проекты без глубоких знаний программирования; музыки и звукового оформления, где ВП используется для генерации звуковых ландшафтов, интерактивных музыкальных инструментов и алгоритмической композиции; театральной сценографии и перформанса с визуальными эффектами и динамическими инсталляциями созданными с помощью ВП.

Тема 2. Языки визуального программирования

Язык визуального программирования – это такой язык, который позволяет программисту создавать программы, манипулируя графическими элементами, а не печатая текстовые команды. Известным примером является Scratch, язык визуального программирования родом из MIT, который используется для обучения детей. Его преимущества заключаются в том, что он делает программирование более доступным для понимания разработчиков не являющихся програмиистами.

Основная идея визуального программирования – это способ образного, графического представления разрабатываемого алгоритма, который наиболее естественен для восприятия человека. ***Очень часто, на практике, это означает визуальное построение интерфейса программы, а не самого кода***. Создаются только заготовки необходимых функций и процедур, а задача создания тела функций по-прежнему – на разработчике программы. В

этом случае мы говорим не о языках визуального программирования а о средах визуального программирования, таких как Visual Basic, Delphi, Borland Builder, Microsoft Visual C++, Visual Studio и др.

Визуальное программирование в качестве объекта визуализации рассматривает процесс построения интерфейса приложения путем размещения на формах компонентов и настройкой их свойств и поведения.

Использование визуального проектирования интерфейса предоставляет программисту следующие преимущества:

– возможность легкого изменения размеров и расположения компонентов на форме (с помощью простых манипуляций мышью);

– в процессе проектирования постоянно виден результат – изображение формы и расположенных на ней компонентов (не надо запускать приложение для проверки внешнего вида окна и последующего изменения программного кода для подбора более удачного размера и расположения компонентов);

– во время проектирования формы и размещения на ней компонентов редактор кода (обычно) автоматически генерирует код программы, включая в нее фрагменты, описывающие данный компонент (далее можно изменять свойства компонентов и писать обработчики событий).

Графические языки программирования

App Inventor – Среда визуальной разработки android-приложений, требующая от пользователя минимальных знаний программирования.

Sketchware – Среда визуальной разработки приложений для Android.

Дракон – графический язык программирования, имеющий корни в программировании ракетно-космической техники («Буран», «Морской старт »). Существуют Дракон-редакторы, включая бесплатные.

Язык последовательных функциональных схем SFC (Sequential Function Chart) – графический язык программирования широко используется для программирования промышленных логических контроллеров PLC. В SFC программа описывается в виде схематической последовательности шагов, объединённых переходами.

HiAsm – это язык и среда разработки приложений, которая позволяет создавать приложения, управляя их моделью с помощью интуитивно понятного графического интерфейса HiAsm.

LD (Ladder Diagram) – язык релейно-контактных схем.

FBD – язык Функциональных блоковых диаграмм.

Язык CFC (Continuous Flow Chart) – высокоуровневый язык графического программирования. CFC — это дальнейшее развития языка FBD. CFC был специально создан для проектирования систем управления непрерывными технологическими процессами.

Язык «G» системы LabVIEW – один из самых распространенных языков разработки программ, работающих с некомпьютерным оборудованием.

VisSim – это визуальный язык программирования предназначенный для моделирования динамических систем, а также проектирования, базирующегося на моделях, для встроенных микропроцессоров.

Блокли – это библиотека для создания среды визуального программирования, которая может быть встроена в произвольное веб-приложение.

Кибор – Интегрированная среда создания бот программ автоматизации. Обладает визуальным инструментом для построения программ с помощью блок схем. Для визуального программирования требуется минимум навыков программирования.

Verge3D Puzzles – основанный на Блокли фреймворк для программирования интерактивных 3D-приложений, работающих в браузере.

Язык визуального программирования (visual programming language), который использует графические компоненты, такие как значки, кнопки и символы как способ разработки IT продуктов. Этот язык программирования позволяет визуально проиллюстрировать программный код. Этот тип языка программирования помогает пользователям без технических знаний в области разработки воспринимать графику и процессы таким образом, который может быть понятен большинству новичков. Визуальный язык программирования также позволяет пользователям просто использовать интерфейс "drag-and-drop" и наиболее эффективно работает на no-code платформах.

Использование визуального языка программирования при разработке программного обеспечения имеет свои преимущества. Современная разработка программного обеспечения на визуальном языке программирования является удобным решением для начинающих пользователей, которые не являются экспертами в области программирования. Простое визуальное расположение изображений и блоков облегчает понимание и проектирование для широкого круга пользователей.

Таким образом, вместо просмотра сложных строк кода, люди могут логически понять и объяснить сложные концепции с помощью визуального языка программирования. Благодаря своей относительной простоте, визуальное программирование является удобным способом ознакомления пользователей с разработкой различных приложений.

Однако, несмотря на простоту языка, он может оказаться сложным к восприятию, поскольку богат различной графикой. Этот язык программирования больше по размеру и поэтому занимает много места на компьютере, что в дальнейшем может привести к замедлению работы некоторых функций из-за объема памяти, который он требует на диске. Языки Визуального программирования очень ограничены в диапазоне функций, которые они используют. Это затрудняет выполнение более сложных операций, и в результате этот тип языка редко используется технологическими гигантами в мире программирования.

Визуальные языки программирования (ВЯП) можно классифицировать по нескольким критериям, в зависимости от их структуры, цели и способа использования:

1. По типу визуального представления:

Блочное программирование использует графические блоки, которые представляют отдельные операции или функции. Блоки соединяются друг с другом для создания программной логики. (Scratch, Blockly, MIT App Inventor)

Диаграммы потоков данных используют диаграммы для визуализации потока данных в программе (LabVIEW, Flowgorithm)

Графические языки программирования используют визуальные элементы, такие как иконки и диаграммы, для представления кода (Visual Programming Language (VPL), VisSim)

Визуальные редакторы кода представляют текстовый код в визуальном формате, делая его более понятным (Visual Studio Code, Atom).

2. По уровню абстракции:

ВЯП с высоким уровнем абстракции предоставляют более простые и интуитивно понятные визуальные элементы, которые отражают абстрактные концепции программирования (Scratch, Blockly)

ВЯП с низким уровнем абстракции предоставляют более детализированные визуальные элементы, которые ближе к машинному коду (LabVIEW, VisSim)

3. По области применения:

Обучение. ВЯП для обучения программированию часто имеют более простой синтаксис и интерфейс, что делает их более доступными для новичков (Scratch, Blockly).

Профессиональная разработка. ВЯП для профессиональной разработки часто имеют более сложный синтаксис и интерфейс, что позволяет создавать более сложные и мощные программы (LabVIEW, VisSim)

Специализированные области. ВЯП для специализированных областей, таких как разработка игр или автоматизация процессов, могут иметь специальные функции и инструменты, которые делают их более эффективными в этих областях.

4. По типу среды выполнения:

Интерпретируемые. Код ВЯП интерпретируется в реальном времени без предварительной компиляции (Scratch, Blockly).

Компилируемые. Код ВЯП компилируется в машинный код перед выполнением (LabVIEW, VisSim)

Языки визуального програмиирвания хорошо зарекомендовали себя в образовании, бизнесе, аналитических сферах, а так же в сфере визуальных искусств.

Образование. Программное обеспечение на языке визуального программирования, которое помогает и направляет процесс обучения, полезно для образования студентов в различных областях. Простые графические компоненты визуального программирования помогают студентам визуализировать и усвоить концепции и процессы быстрее. Платформы визуального языка программирования и их интерфейс настолько просты для восприятия, что студенты могут изучать основы программирования на этих платформах и создавать свои приложения. No-code платформы помогают пользователям без технических знаний быстрее и легче выйти на рынок ИТ.

Визуальное моделирование приложений

В современной разработке программного обеспечения визуальный язык программирования – это программное обеспечение, которое помогает иллюстрировать логические концепции и процессы блок-схемами. Например, визуальные языки программирования широко используются в визуальном моделировании приложений для имитации особенностей прототипов. Этот

процесс разработки программного обеспечения, используемый многими организациями, известен как Visual App-Modeling.

Приложения для внутреннего использования

Реальные приложения созданные с помощью языка визуального программирования включают управление данными, бизнес-процессами и аналитику. Визуальный язык программирования помогает компаниям масштабироваться благодаря интуитивно понятному интерфейсу и простому восприятию автоматизированных процессов. Он заменяет необходимость в высокооплачиваемых программистах, позволяя бизнесу собирать, создавать и генерировать ценные данные, отчеты и аналитику.

Тема 6. Разработка приложений в средах визуального программирования

Интегрированная среда разработки – это совокупность программных средств, поддерживающая все этапы разработки программного обеспечения от написания исходного текста программы до ее компиляции и отладки, и обеспечивающая простое и быстрое взаимодействие с другими инструментальными средствами (программным отладчиком-симулятором, внутрисхемным эмулятором, эмулятором и программатором).

Строго говоря, интегрированные среды разработки не относятся к числу средств отладки, тем не менее обойти вниманием данный класс программных средств, существенно облегчающий и ускоряющий процесс разработки и отладки микропроцессорных систем было бы неправильно.

При традиционном подходе, начальный этап написания программы строится следующим образом:

1. Исходный текст набирается при помощи какого-либо текстового редактора. По завершении набора, работа с текстовым редактором прекращается и запускается кросс компилятор. Как правило, вновь написанная программа содержит синтаксические ошибки, и компилятор сообщает о них на консоль оператора.

2. Вновь запускается текстовый редактор, и оператор должен найти и устранить выявленные ошибки, при этом сообщения о характере ошибок выведенные компилятором уже не видны, так как экран занят текстовым редактором.

И этот цикл может повторяться не один раз. Если программа иммет большой объем, собирается из различных частей, и подвергается длительному редактированию или модернизации, то даже этот начальный этап может потребовать много сил и времени. После этого наступает этап отладки программы и к редактору с компилятором добавляется эмулятор или симулятор, за работой которого хотелось бы следить прямо по тексту программы в текстовом редакторе.

Избежать большого объема однообразных децствий и тем самым существенно повысить эффективность процесса разработки и отладки позволяют т.н. интегрированные среды (оболочки) разработки (Integrated Development Environment, IDE).

Работа в интегрированной среде дает программисту:

– Возможность использования встроенного многофайлового текстового редактора, специально ориентированного на работу с исходными текстами программ;

– Диагностика выявленных при компиляции ошибок, и исходный текст программы, доступный редактированию, выводятся одновременно в многооконном режиме;

– Возможность организации и ведения параллельной работы над несколькими проектами. Менеджер проектов позволяет использовать любой проект в качестве шаблона для вновь создаваемого проекта;

– Перекомпиляции подвергаются только редактировавшиеся модули;

– Возможность загрузки отлаживаемой программы в имеющиеся средства отладки, и работы с ними без выхода из оболочки;

– Возможность подключения к оболочке практически любых программных средств.

В последнее время, функции интегрированных сред разработки становятся стандартной принадлежностью программных интерфейсов эмуляторов и отладчиков-симуляторов.

Подобные функциональные возможности, в сочетании с дружественным интерфейсом, в состоянии существенно увеличить скорость разработки программ для микроконтролленров и процессоров цифровой обработки сигналов.

Решаемая на ПК задача реализуется в виде прикладной программы – приложения. В основе разработки приложения лежит ***проект***. Центральная

часть проекта – форма, на нее помещаются необходимые для решения конкретной задачи компоненты.

Проект – результат процессов программирования и проектирования, объединяет программный код и графический интерфейс. Включает программные модули форм и самостоятельные программные модули в виде отдельных файлов. Проект может быть запущен на выполнение только из среды разработки.

Событие – изменение некоторого состояния, распознаваемое объектом. Для реакции на это изменение могут быть описаны некоторые методы - обработчики, обрабатывающие события в программном коде.

Обработчик события – процедура, которая начинает выполняться после реализации определенного события (щелчка).

Приложение интегрирует программный код и графический интерфейс в одном исполняемом файле, он может запускаться непосредственно в ОС.

Проект работает только из среды разработки и состоит из нескольких файлов, а приложение работает из ОС и состоит (обычно) из одного файла

Этапы создания проектов и приложений в средах визуального программирования:

Создание графического интерфейса проекта. На форму помещаются элементы управления, обеспечивающие взаимодействие проекта с пользователем.

Установка значений свойств объектов графического интерфейса. В режиме конструирования задаются значения свойств формы и элементов управления, помещенных ранее на форму.

Создание и редактирование программного кода. Создаются заготовки обработчиков событий (двойной щелчок мышью по элементу - вызов заготовки обработчика наиболее часто используемого события для этого элемента). В редакторе программного кода производится ввод и редактирование программного кода обработчиков событий.

Сохранение проекта. Так как проекты включают в себя несколько файлов, рекомендуется для каждого проекта создать отдельную папку на диске.

Компиляция проекта в приложение. Сохраненный проект может выполняться только в самой системе программирования. Чтобы преобразовать проект в приложение, которое может выполняться

непосредственно в среде ОС, необходимо выполнить компиляцию проекта, в процессе которой приложение сохранится в исполнимом файле (.exe).

Таким образом, приложение собирается из элементов: форм, программных модулей, внешних библиотек, картинок, пиктограмм и др. Каждый элемент размещается в отдельном файле и имеет строго определенное назначение. Набор всех файлов, необходимых для создания приложения, называется проектом. Компилятор последовательно обрабатывает файлы проекта и строит из них выполняемый файл.

Типы основных файлов проекта:

Файлы описания форм. Текстовый файл, описывающий форму с компонентами. В нем сохраняются значения свойств формы и ее компонентов, установленные в окне свойств во время проектирования приложения. Количество файлов равно количеству используемых в приложении форм (если используется только одна форма, то файл только один).

Файлы программных модулей. Текстовые файлы, содержащие исходные программные коды. В них пишут методы обработки событий, генерируемых формами и компонентами. Каждой форме в проекте соответствует свой программный модуль (unit), содержащий все относящиеся к форме объявления и методы обработки событий, написанные на языке программирования.

Программные модули размещаются в отдельных файлах. Их количество может превышать количество форм, так как программные модули могут и не относиться к формам, а содержать вспомогательные процедуры, функции, классы и др.

Главный файл проекта. Текстовый файл, содержащий главный программный блок. Чтобы компилятор знал, какие конкретно файлы входят в проект, необходим главный программный файл, который подключает все файлы модулей, входящих в проект. Для каждого проекта существует только один такой файл.

Как правило по команде подобной "File / New / Application" начинается разработка нового приложения, среда автоматически создает файл проекта. По мере создания новых форм содержимое этого файла видоизменяется автоматически. После окончания работы в файле будет находиться перечень программных модулей, которые будут поданы на вход компилятору. В редакторе кода появится новая страница с текстом.

В проект могут входить логически автономные элементы: точечные рисунки (BMP-файлы), значки (ICO-файлы), файлы справки (HLP-файлы) и т.п., но ими управляет сам программист.

Файл проекта подключает все используемые программные модули и содержит операторы для запуска приложения. Этот файл среда разработки создает и контролирует сама.

Класс – в объектно-ориентированном программировании (ООП), модель для создания объектов определённого типа, описывающая их структуру (набор полей и их начальное состояние) и определяющая алгоритмы (функции или методы) для работы с этими объектами.

Иными словами, класс служит средством для введения абстрактных типов данных в программный проект. Другие описатели абстрактных типов данных – метаклассы, интерфейсы, структуры, перечисления, — характеризуются какими-то своими особенностями. Суть отличия классов состоит в том, что при задании типа данных, класс определяет одновременно как интерфейс, так и реализацию для всех своих экземпляров (т.е. объектов), поэтому вызов метода-конструктора обязателен.

Класс является одним из ключевых понятий в ООП, но существуют и бесклассовые объектно-ориентированные языки, например, Self, JavaScript, Lua, (подробнее смотрите Прототипное программирование).

На практике объектно-ориентированное программирование сводится к созданию некоторого количества классов, включая интерфейс и реализацию, и последующему их использованию. Графическое представление некоторого количества классов и связей между ними называется диаграммой классов. Объектно-ориентированный подход за время своего развития накопил множество рекомендаций (паттернов) по созданию классов и иерархий классов.

Идея классов пришла из работ по базам знаний, имеющих отношение к исследованиям по искусственному интеллекту. Используемые человеком классификации в зоологии, ботанике, химии, деталях машин, несут в себе основную идею, что любую вещь всегда можно представить частным случаем некоторого более общего понятия.

Классы и объекты, понятие экземпляра класса, понятие членов класса

В объектно-ориентированной программе с применением классов каждый объект является «экземпляром» некоторого конкретного класса, и

других объектов не предусмотрено. То есть «экземпляр класса» в данном случае означает не «пример некоторого класса» или «отдельно взятый класс», а «объект, типом которого является какой-то класс». При этом в разных языках программирования допускается либо не допускается существование еще каких-то типов данных, экземпляры которых не являются объектами (то есть язык определяет, являются ли объектами такие вещи, как числа, массивы и указатели, или не являются, и, соответственно, есть ли такие классы как «число», «массив» или «указатель», экземплярами которых были бы каждое конкретное число, массив или указатель).

Например, абстрактный тип данных «строка текста» может быть оформлен в виде класса, и тогда все строки текста в программе будут являться объектами – экземплярами класса «строка текста».

При использовании классов все элементы кода программы, такие как переменные, константы, методы, процедуры и функции, могут принадлежать (а во многих языках обязаны принадлежать) тому или иному классу. Сам класс в итоге определяется как список своих членов, а именно полей (свойств) и методов/функций/процедур. В зависимости от языка программирования к этому списку могут добавиться константы, атрибуты и внешние определения.

Как и структуры, классы могут задавать поля – то есть переменные, принадлежащие либо непосредственно самому классу (статические), либо экземплярам класса (обычные). Статические поля существуют в одном экземпляре на всю программу (или, в более сложном варианте, – в одном экземпляре на процесс или на поток/нить). Обычные поля создаются по одной копии для каждого конкретного объекта — экземпляра класса. Например, общее количество строк текста, созданных в программе за время её работы, будет являться статическим полем класса «строка текста». А конкретный массив символов строки будет являться обычным полем экземпляра класса «строка текста», так же как переменная «фамилия», имеющая тип «строка текста», будет являться обычным полем каждого конкретного экземпляра класса «человек».

В ООП при использовании классов весь исполняемый код программы (алгоритмы) будет оформляться в виде так называемых «методов», «функций» или «процедур», что соответствует обычному структурному программированию, однако теперь они могут (а во многих языках обязаны) принадлежать тому или иному классу. Например, по возможности, класс

«строка текста» будет содержать все основные методы/функции/процедуры, предназначенные для работы со строкой текста, такие как поиск в строке, вырезание части строки и т. д.

Как и поля, код в виде методов/функций/процедур, принадлежащих классу, может быть отнесен либо к самому классу, либо к экземплярам класса. Метод, принадлежащий классу и соотнесенный с классом (статический метод) может быть вызван сам по себе и имеет доступ к статическим переменным класса. Метод, соотнесенный с экземпляром класса (обычный метод), может быть вызван только у самого объекта, и имеет доступ как к статическим полям класса, так и к обычным полям конкретного объекта (при вызове этот объект передастся скрытым параметром метода). Например, общее количество созданных строк можно узнать из любого места программы, но длину конкретной строки можно узнать только указав, тем или иным образом, длину какой строки будем мерить.

Интерфейс и реализация, наследование реализации

В программировании есть понятие программного интерфейса, означающего перечень возможных вычислений, которые может выполнить та или иная часть программы. Это включает описание: какие аргументы и в каком порядке требуется передавать на вход алгоритмам из перечня, а также что и в каком виде они будут возвращать. Абстрактный тип данных интерфейс придуман для формализованного описания такого перечня. Сами алгоритмы, то есть действительный программный код, который будет выполнять все эти вычисления, интерфейсом не задаётся, программируется отдельно и называется реализацией интерфейса.

Программные интерфейсы, а также классы, могут расширяться путём наследования, которое является одним из важных средств повторного использования готового кода в ООП. Наследованный класс или интерфейс будет содержать в себе всё, что указано для всех его родительских классов (в зависимости от языка программирования и платформы, их может быть от нуля до бесконечности). Например, можно создать свой вариант текстовой строки путём наследования класса «моя строка текста» от уже существующего класса «строка текста», при этом предполагается, что программисту не придется заново переписывать алгоритмы поиска и прочее, так как они автоматически будут унаследованы от готового класса, и любой экземпляр класса «моя строка текста» может быть передан не только в готовые методы родительского класса «строка текста» для проведения

нужных вычислений, но и вообще в любой алгоритм, способный работать с объектами типа «строка текста», так как экземпляры обоих классов совместимы по программным интерфейсам.

Класс позволяет задать не только программный интерфейс к самому себе и к своим экземплярам, но и в явном виде написать код, ответственный за вычисления. Если при создании своего нового типа данных наследовать интерфейс, то мы получим возможность передавать экземпляр своего типа данных в любой алгоритм, который умеет работать с этим интерфейсом. Однако нам придется самим написать реализацию интерфейса, то есть те алгоритмы, которыми будет пользоваться интересующий нас алгоритм для проведения вычислений с использованием нашего экземпляра. В то же время, наследуя класс, мы автоматически наследуем готовый код под интерфейс (это не всегда так, родительский класс может требовать реализации каких-то алгоритмов в дочернем классе в обязательном порядке). В этой возможности наследовать готовый код и проявляется то, что в объектно-ориентированной программе тип данных класс определяет одновременно и интерфейс, и реализацию для всех своих экземпляров.

Состояние объекта, понятие областей доступа, конструкторы

Одной из проблем структурного программирования, с которой борется ООП, является проблема поддержания правильного значения переменных программы. Часто разные переменные программы хранят логически связанные значения, и за поддержание этой логической связности несет ответственность программист, то есть автоматически связность не поддерживается. Примером могут служить флажки «уволен» и «ожидает премии по итогам года», когда по правилам отдела кадров человек может быть одновременно не уволенным и не ожидающим премии, не уволенным и ожидающим премии, уволенным и не ожидающим премии, но не может быть одновременно и уволенным, и ожидающим премии. То есть любая часть программы, которая проставляет флажок «уволен», всегда должна снимать флажок «ожидает премии по итогам года».

Хороший способ решить эту проблему – объявить флажок «уволен» недоступным к изменению для всех участков программы, кроме одного специально оговоренного. В этом специально оговоренном участке всё будет написано один раз и правильно, а все остальные должны будут обращаться к этому участку всегда, когда они хотят установить или снять флажок «уволен».

В объектно-ориентированной программе флажок «уволен» будет объявлен приватным членом некоторого класса, а для чтения и изменения его будут написаны соответствующие публичные методы. Правила, определяющие возможность или невозможность напрямую изменять какие-либо переменные, называются правилами задания областей доступа. Слова «приватный» и «публичный» в данном случае являются так называемыми «модификаторами доступа». Они называются модификаторами потому, что в некоторых языках они используются для изменения ранее установленных прав при наследовании класса. Совместно классы и модификаторы доступа задают область доступа, то есть у каждого участка кода, в зависимости от того, какому классу он принадлежит, будет своя область доступа относительно тех или иных элементов (членов) своего класса и других классов, включая переменные, методы, функции, константы и т. д. Существует основное правило: ничто в одном классе не может видеть приватных элементов другого класса. Относительно других, более сложных правил, в различных языках существуют другие модификаторы доступа и правила их взаимодействия с классами.

Почти каждому члену класса можно установить модификатор доступа (за исключением статических конструкторов и некоторых других вещей). В большинстве объектно-ориентированных языков программирования поддерживаются следующие модификаторы доступа:

private (закрытый, внутренний член класса) – обращения к члену допускаются только из методов того класса, в котором этот член определён. Любые наследники класса уже не смогут получить доступ к этому члену. Наследование по типу private делает все члены родительского класса (в том числе public и protected) private-членами класса-наследника;

protected (защищённый, внутренний член иерархии классов) – обращения к члену допускаются из методов того класса, в котором этот член определён, а также из любых методов его классов-наследников. Наследование по типу protected делает все public-члены родительского класса protected-членами класса-наследника;

public (открытый член класса) – обращения к члену допускаются из любой части кода. Наследование по типу public не меняет модификаторов родительского класса;

Проблема поддержания правильного состояния переменных актуальна и для самого первого момента выставления начальных значений. Для этого в

классах предусмотрены специальные методы/функции, называемые конструкторами. Ни один объект (экземпляр класса) не может быть создан иначе, как путём вызова на исполнение кода конструктора, который вернет вызывающей стороне созданный и правильно заполненный экземпляр класса. Во многих языках программирования тип данных «структура», как и класс, может содержать переменные и методы, но экземпляры структур, оставаясь просто размеченным участком оперативной памяти, могут создаваться в обход конструкторов, что запрещено для экземпляров классов (за исключением специальных исключительных методов обхода всех подобных правил ООП, предусмотренных в некоторых языках и платформах). В этом проявляется отличие классов от других типов данных – вызов конструктора обязателен.

Наследование (англ. inheritance) – концепция объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.

В объектно-ориентированном программировании, абстрактные типы данных называются классами.

Суперкласс (англ. super class), родительский класс (англ. parent class), предок, родитель или надкласс – класс, производящий наследование в подклассах, т. е. класс, от которого наследуются другие классы. Суперклассом может быть подкласс, базовый класс, абстрактный класс и интерфейс.

Подкласс (англ. subclass), производный класс (англ. derived class), дочерний класс (англ. child class), класс потомок, класс наследник или класс-реализатор – класс, наследуемый от суперкласса или интерфейса, т. е. класс определённый через наследование от другого класса или нескольких таких классов. Подклассом может быть суперкласс.

Базовый класс (англ. base class) – это класс, находящийся на вершине иерархии наследования классов и в основании дерева подклассов, т. е. не являющийся подклассом и не имеющий наследований от других суперклассов или интерфейсов. Базовым классом может быть абстрактный класс и интерфейс. Любой не базовый класс является подклассом.

Интерфейс (англ. interface) – это структура, определяющая чистый интерфейс класса, состоящий из абстрактных методов. Интерфейсы участвуют в иерархии наследований классов и интерфейсов.

Суперинтерфейс (англ. super interface) или интерфейс-предок – это аналог суперкласса в иерархии наследований, т. е. это интерфейс производящий наследование в подклассах и подинтерфейсах.

Интерфейс-потомок, интерфейс-наследник или производный интерфейс (англ. derived interface) — это аналог подкласса в иерархии наследований интерфейсов, т. е. это интерфейс наследуемый от одного или нескольких суперинтерфейсов.

Базовый интерфейс – это аналог базового класса в иерархии наследований интерфейсов, т. е. это интерфейс, находящийся на вершине иерархии наследования.

Иерархия наследования или иерархия классов – дерево, элементами которого являются классы и интерфейсы.

Наследование является механизмом повторного использования кода (англ. code reuse) и способствует независимому расширению программного обеспечения через открытые классы (англ. public classes) и интерфейсы (англ. interfaces). Установка отношения наследования между классами порождает иерархию классов (англ. class hierarchy).

Простое наследование, иногда называемое одиночным наследованием, описывает родство между двумя классами: один из которых наследует второму. Из одного класса могут выводиться многие классы, но даже в этом случае подобный вид взаимосвязи остается «простым» наследованием.

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++. Из других языков, предоставляющих эту возможность, можно отметить Python и Eiffel. Множественное наследование поддерживается в языке UML.

Множественное наследование – потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имён методов в предках. В языках, которые позиционируются как наследники C++ (Java, C# и другие), от множественного наследования было решено отказаться в пользу интерфейсов. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость всё-таки возникла, то

для разрешения конфликтов использования наследованных методов с одинаковыми именами возможно, например, применить операцию расширения видимости – «::» – для вызова конкретного метода конкретного родителя.

Попытка решения проблемы наличия одинаковых имён методов в предках была предпринята в языке Eiffel, в котором при описании нового класса необходимо явно указывать импортируемые члены каждого из наследуемых классов и их именование в дочернем классе.

Большинство современных объектно-ориентированных языков программирования (C#, Java, Delphi и другие) 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.

МАТЕРИАЛЫ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

Тема 1. Введение в визуальное программирование

Лабораторная работа №1
Графическое представление потока данных между компонентами системы (2 часа)

Цель: ***научиться применять графическое представление потока данных между компонентами системы в программирования и разработки мобильных приложений***

Задание 1. Создать проект игры викторины для мобильного приложения.

Задание 2. Создать UML диаграмму, отражающую процесс взаимодействия пользователя и приложения.

Задача 3. Создать блочные диаграммы алгоритмов программирования нескольких компонент приложения (кнопок, текстовых полей и т.д.)

Тема 3. Визуальные среды разработки программ

Лабораторная работа 1
Изучение визуальные среды разработки программ с использованием языков визуального (графического) программирования (2 часа)

Цель: изучить интерфейс визуальной среды разработки программ с использованием языков визуального (графического) программирования

Задание 1. Изучить алгоритмические конструкции: 1) следование, условие, повторение; 2) циклы: с известным числом повторений, с предусловием и постусловием, безусловные циклы; счетчики;

Задание 2. Изучить работу с графическими объектами в визуальной среде.

Задание 3. Изучить: 1) работу с компонентами: свойства, события, методы; 2) основные свойства и методы формы, и организацию взаимодействия форм; 3) компоненты объединения элементов управления (контейнеры, область прокрутки, фреймы); основные компоненты управления (кнопки, простой список и комбинированный список, флажки и переключатели).

Тема 4. Визуальные среды для разработки мобильных приложений и игр

Лабораторная работа №1
Проект игрового мобильного приложения (2 часа)

Цель: научиться разрабатывать проект и визуальную реализацию мобильного игрового приложения для сферы культуры

Задание 1. Разработать концепцию и проект игрового мобильного приложения-викторины для сферы социально-культурной деятельности.

Задание 2. Создать визуального решения мобильного приложения. Спроектировать действия компонент приложения.

Лабораторная работа №2 (2 часа)
Реализация игрового мобильного приложения в визуальной среде для разработки мобильных приложений и игр

Задание 1. Реализовать интерфейс мобильного приложения-викторины.

Задание 2 Запрограммировать действия структурных компонент приложения-викторины.

Задание 3. Осуществить тестирование мобильного приложения-викторины.

Тема 5. Визуальное программирование для создания цифровых 2D и 3D объектов сферы социально-культурной деятельности

Лабораторная работа №1
Проект графического продукта для сферы социально-культурной деятельности. (4 часа)

Цель: научиться разрабатывать проект и визуальную реализацию мобильного игрового приложения для сферы культуры

Задание 1. Разработать концепцию графического продукта для сферы социально-культурной деятельности.

Задание 2. Спроектировать компоненты, требующие программных решений и конвертировать логику алгоритмов действия компонент в логику визуального скриптинга.

Задание 3. Итогово реализовать графический проект. Создать и протестировать визуальный скриптинг. Продемонстрировать графический продукт для сферы социально-культурной деятельности.

Тема 6. Разработка приложений в средах визуального програмирования

Лабораторная работа №1
Создание пользоательских форм в приложениях приложениях Office (4 часа)

Цель: научиться создавать пользовательские формы для приложений Office

Справочная Информация

Практически во всех приложениях Office используются пользовательские диалоговые окна. Диалоговые окна в VBA называются формами (объект UserForms). Каждому объекту UserForm присущи определенные свойства, методы и события, которые он наследует от класса объектов UserForms. Диалоговые окна (формы) и элементы управления составляют основу современного визуального интерфейса. Все элементы управления и технология работы с ними в основном стандартизованы и похожи для разных платформ и программных сред. Эти объекты помещены в специальную библиотеку MSForms. Выделим основные моменты, которые следует иметь в виду при создании визуального интерфейса.

Все загруженные диалоговые окна представляют коллекцию UserForms со стандартными методами и свойствами. Элемент коллекции – объект класса UserForm – задает отдельное окно. Для каждого типа элементов управления в библиотеке msforms имеется класс объектов, имя которого совпадает с именем элемента управления (его типа). Например, есть классы SpinButton и TextBox.

Диалоговые окна создаются, как правило, не программно, а визуально. Вначале создается само окно, а затем оно наполняется элементами управления при помощи соответствующей панели элементов. Этот этап называется этапом проектирования, и его следует отличать от этапа выполнения, когда приложение выполняется и конечный пользователь взаимодействует с приложением, в частности через диалоговые окна и их элементы управления. Как только создается диалоговое окно и помещается в него тот или иной элемент управления, в этот же самый момент автоматически в программе появляется объект соответствующего класса, с которым можно работать, вызывая его методы и изменяя его свойства. На этапе проектирования, используя окно свойств, можно задать большинство свойств, как самого диалогового окна, так и всех элементов управления, помещенных в него, кроме этого, программно необходимо прописать все обработчики событий.

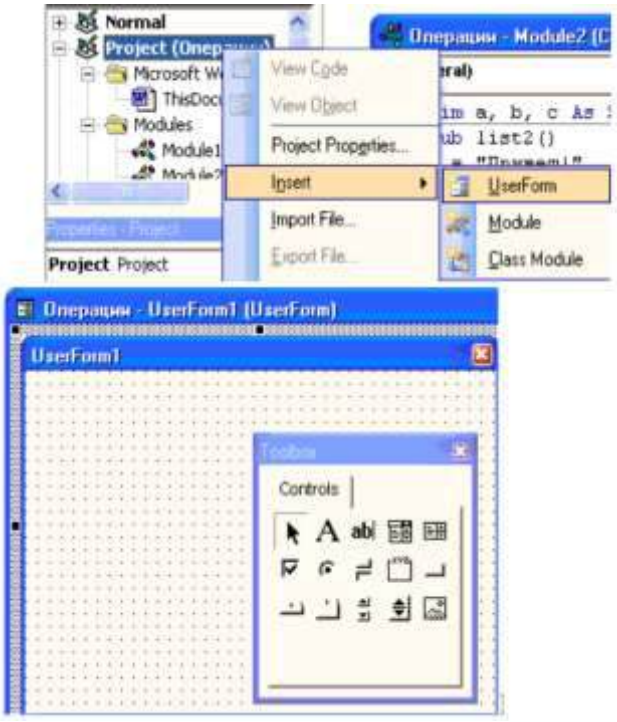


Рисунок 3 – Описание событий

Последний момент – отладка. Для ведения отладки нужно предварительно откомпилировать приложение и затем перейти в режим выполнения приложения. Для того чтобы в разрабатываемое приложение можно было добавить форму, необходимо выполнить следующие действия: запустить редактор VBA; выделить правой кнопкой мыши объект Project, выполнить команду Insert + UserForm, после чего появляются новая форма и панель элементов Toolbox.

Форма как объект имеет некоторые встроенные свойства, и их можно устанавливать или программным образом, или в Properties Window (окне свойств) редактора VBA (таблица. Наиболее часто используемые свойства объектов UserForm)

Наиболее часто используемые свойства объектов UserForm

Свойство	Описание
ActiveControl	Возвращает объектную ссылку на элемент управления, находящийся в фокусе в данный момент. Только для чтения
BackColor	Целое типа Long определяет цвет фона формы
Caption	Текст, выводимый в качестве заголовка формы
Controls	Возвращает коллекцию всех элементов управления формы
Cycle	Определяет, должно ли нажатие клавиши табуляции вызывать последовательный выбор всех элементов управления во всех группах и на каждой странице многостраничных элементов управления или только в пределах текущей группы или страницы. Может содержать одну из двух встроенных констант: fmCycleAllForms или fmCycleCurrentForm
Enabled	Содержит значение типа Boolean, указывающее, доступна ли форма. Если его значение равно False, ни один из элементов управления формы не доступен
Font	Возвращает ссылку на объект Font, посредством которого можно выбрать параметры шрифта формы или элемента управления
ForeColor	То же самое, что и свойство BackColor, но устанавливает цвет, используемый для переднего плана (обычно это цвет текста) объекта формы

Наиболее часто используемые методы для объектов UserForm

Метод	Назначение
Copy	Копирует выделенный в элементе управления текст в буфер обмена Windows
Cut	Вырезает выделенный в элементе управления текст и помещает его в буфер обмена Windows
Hide	Скрывает UserForm, не выгружая ее из памяти, сохраняя значения элементов управления формы и всех переменных, объявленных в модуле класса формы
Paste	Вставляет содержимое буфера обмена Windows в текущий элемент управления
PrintForm	Выводит на используемый в Windows по умолчанию принтер изображение формы, включая все данные, введенные в элементы управления
Repaint	Перерисовывает форму, выведенную на экран. Используется этот метод, если необходимо перерисовать форму, не ожидая, когда она будет перерисована через обычный период времени
Show	Выводит форму на экран. Если форма еще не загружена в память, то данный метод сначала ее загружает. Синтаксис метода Show:FormName.Show

События объектов UserForm

Событие	Синтаксис заголовка процедуры	Описание
Activate	Private Sub object_Activate()	Инициируется всякий раз, когда окно формы становится активным. Используйте это событие для обновления содержимого диалоговых элементов управления, чтобы отразить любые изменения, которые произошли, пока окно было неактивным
Click	Private Sub object_Click()	Инициируется всякий раз, когда по форме (любой ее части, не занятой элементами управления) щелкают мышью
DblClick	Private Sub object_DblClick()	Инициируется всякий раз, когда по форме (любой ее части, не занятой элементами управления) дважды щелкают мышью
Deactivate	Private Sub object_Deactivate()	Инициируется всякий раз, когда форма перестает быть активной
Initialize	Private Sub object_Initialize()	Инициируется всякий раз, когда форма впервые загружается в память посредством выполнения оператора Load или с помощью метода Show. Используйте это событие для инициализации элементов управления формы при ее появлении на экране
Resize	Private Sub object_Resize()	Инициируется при изменении размеров формы
Terminate	Private Sub object_ Terminate()	Инициируется всякий раз, когда форма выгружается из памяти. Используйте это событие для осуществления любых специальных служебных задач, которые необходимо выполнить прежде, чем переменные формы будут выгружены

Объект UserForm может содержать те же элементы управления, что и находящиеся в диалоговых окнах Word, Excel или других приложений Windows (таблица. Стандартные элементы управления, включенные в VBA). Элементы управления – это элементы диалогового окна, позволяющие пользователю взаимодействовать с программой. Они включают в себя кнопки-переключатели, текстовые поля, линейки прокрутки, командные кнопки и так далее.

Стандартные элементы управления, включенные в VBA

Элемент управления	Назначение
Label (надпись, метка)	Позволяет создавать заголовки элементов управления, которые не имеют собственных встроенных заголовков
TextBox (текстовое поле)	Окно редактируемого текста свободной формы для ввода данных. Может быть одно- или многострочным
ComboBox (поле со списком)	Этот элемент управления объединяет окно редактирования и окно списка. Используйте, когда хотите предложить пользователю выбрать значение, но при этом дать ему возможность ввести данные, отсутствующие в списке
ListBox (список)	Отображает список значений, из которых пользователь может сделать выбор. Окна списка можно использовать, чтобы дать возможность пользователю выбрать только одно значение или же несколько
CheckBox (флажок)	Стандартный флажок (квадратное окно, содержащее, если элемент выбран, галочку). Используйте флажки для выбора вариантов, которые не являются взаимоисключающими
OptionButton (переключатель)	Стандартная кнопка-переключатель (круглое окно, при выборе в центре него находится черная точка). Используйте OptionButton, когда пользователю необходимо сделать выбор между положениями «включено/выключено» или «истина/ложь». Кнопки-переключатели, как правило, объединяются вместе при помощи рамки для создания группы переключателей
ToggleButton (выключатель)	Выключатели служат для той же цели, что и флажки, но выводят установки в виде кнопки, находящейся в «нажатом» или «отжатом» состоянии
Frame (рамка)	Визуально и логически объединяет некоторые элементы управления (особенно флажки, переключатели и выключатели)
CommandButton (кнопка)	Используйте кнопки для выполнения таких действий, как Cancel (Отмена), Save (Сохранить), OK и так далее Когда пользователь щелкает по кнопке, выполняется VBA-процедура, закрепленная за данным элементом управления
TabStrip (набор вкладок)	Этот элемент управления состоит из области, в которую вы помещаете другие элементы управления (такие как текстовые поля, флажки и так далее) и полосы кнопок табуляции. Используйте элемент управления TabStrip для создания диалоговых вкладок, отображающих одни и те же данные в различных категориях
MultiPage (набор страниц)	Этот элемент управления состоит из нескольких страниц. Вы можете выбрать любую из них, щелкнув по соответствующей вкладке. Используйте элемент управления MultiPage для создания диалоговых окон с вкладками
ScrollBar (полоса прокрутки) и SpinButton (счетчик)	Элемент управления ScrollBar позволяет выбирать линейное значение аналогично тому, как это можно сделать при помощи счетчика. Элемент управления SpinButton является специальной разновидностью текстового поля
Image (рисунок)	Элемент управления Image позволяет вывести на форме графическое изображение. Используйте Image для вывода графических изображений в любом из следующих форматов: *.bmp, *.cur, *.gif, *.ico, *.jpg или *.wmf

Свойства стандартных элементов управления

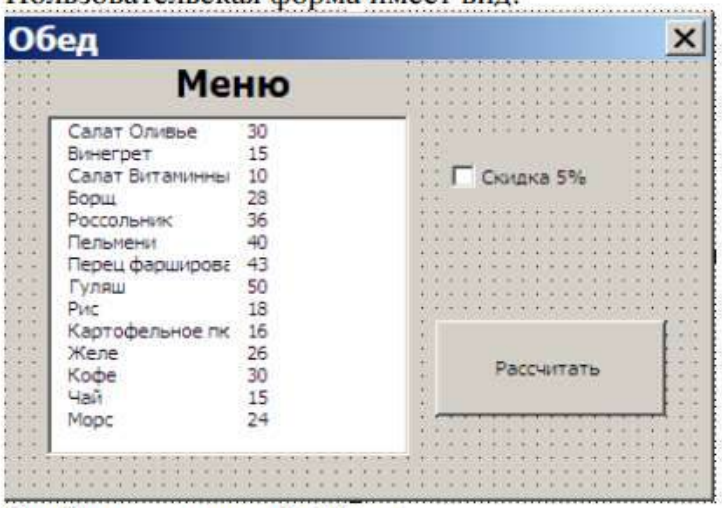
Свойство	Где применяется	Описание
Accelerator	CheckBox, Tab, CommandButton, Label, Page, OptionButton, ToggleButton	Содержит символ, используемый в качестве быстрой клавиши вызова, элемента управления, при нажатии Alt + <клавиша быстрого вызова> происходит выбор элемента управления
BackColor	Все элементы	Число, представляющее определенный цвет фона элемента управления
Caption	CheckBox, CommandButton,	Для надписи - текст, отображаемый элементом управления. Для других
	Frame, Label, OptionButton, ToggleButton, Page, Tab, UserForm	элементов управления - надпись, которая появляется на кнопке или вкладке или рядом с рамкой, флажком или переключателем
Cancel	CommandButton	Задает кнопку отмены диалогового окна. При нажатии на эту кнопку или клавишу Esc диалоговое окно исчезает. Только одна кнопка формы может иметь данное свойство
ControlTip-Text	Все элементы управления	Устанавливает текст, который отображается в виде всплывающей подсказки (ControlTip, называемой также ToolTip), когда указатель мыши помещается на элемент управления
Default	CommandButton	Определяет заданную по умолчанию кнопку. Когда пользователь нажимает в процессе диалога клавишу Enter, эта кнопка ведет себя так, как если бы по ней щелкнули мышью
Enabled	Все элементы управления	Хранит значение типа Boolean, определяющее, доступен или нет элемент управления. Если Enabled имеет значение False, то элемент управления продолжает отображаться в диалоговом окне, но не может быть выбран
ForeColor	Все элементы управления	То же самое, что и BackColor, но устанавливает цвет для переднего плана элемента управления, как правило, символов текста
List	ComboBox	Массив типа variant (одно- или многомерный), представляет список, содержащийся в элементе управления
Max	ScrollBar, SpinButton	Переменная типа Long, определяющая максимальное значение счетчика, или значение, при котором полоса прокрутки находится в самом верху (для вертикальной полосы) или справа (для горизонтальной)

Min	ScrollBar, SpinButton	Переменная типа Long, определяющая минимальное значение счетчика, или значение, при котором полоса прокрутки находится в самом низу (для вертикальной полосы) или слева (для горизонтальной)
Name	Все элементы управления	Содержит имя элемента управления. Вы можете установить данное свойство только с помощью Properties Window
RowSource	ComboBox	Задает источник, из которого ListBox берет список объекта. В Excel VBA RowSource обычно использует диапазон рабочего листа
Selected	ListBox	Возвращает массив значений типа Boolean для списка, который допускает множественный выбор. Каждый элемент массива содержит по одному элементу, соответствующему каждому пункту списка. Если значение элемента в массиве selected равно True, то соответствующий пункт списка выбран
TabIndex	Все элементы управления	Число, указывающее положение элемента управления в порядке табуляции (может иметь значение от 0 до значения, равного количеству элементов управления на форме)
TabStop	Все элементы управления	Значение типа Boolean, указывающее, может ли элемент управления быть выбран клавишей Tab. Если значение TabStop равно False, вы тем не менее можете щелкнуть на элементе и таким образом его выбрать
Value	Все элементы управления	Значение текущих установок элемента управления: текст в текстовом поле, какие выбраны флажки и переключатели, индекс выбранного раздела списка или число, указывающее текущее положение полосы прокрутки или счетчика
Visible	Все элементы управления	Значение типа Boolean, указывающее, является ли элемент управления видимым

Задание 1. Создать приложение Меню для выбора блюд из списка и вывода результата выбора на лист Excel. Можно одновременно выбрать несколько 113 блюд. Предусмотрено наличие скидки 5%. Данные о блюдах (название блюда и его цена) находятся на листе Excel с именем «списки».

	A	B	C
1	Салат Оливье	30	
2	Винегрет	15	
3	Салат Витаминный	10	
4	Борщ	28	
5	Россольник	36	
6	Пельмени	40	
7	Перец фаршированный	43	
8	Гуляш	50	
9	Рис	18	
10	Картофельное пюре	16	
11	Желе	26	
12	Кофе	30	
13	Чай	15	
14	Морс	24	
15			

Пользовательская форма имеет вид:



Свойства элемента ListBox1:

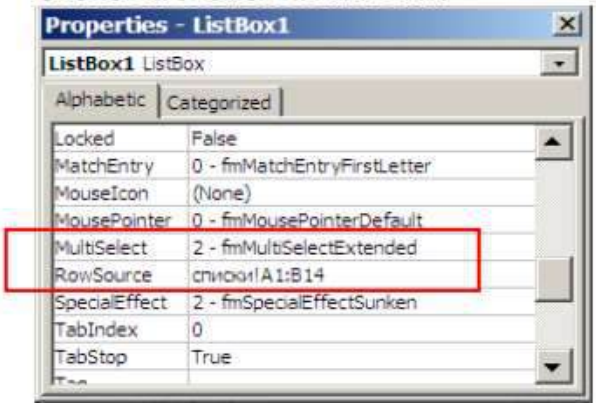


Рисунок 4 – Иллюстрация к заданию 1.1

Задание 2. Создать приложение, рассчитывающее стоимость железнодорожного билета в зависимости от направления вида вагона и сезона. Летом стоимость увеличивается на 20%, зимой - уменьшается на 10%. Данные о стоимости билетов по-прежнему располагаются на листе Excel. Список заполняется при инициализации формы. Количество элементов в списке определяются во время работы программы, а не во время создания формы, как это было в предыдущем задании.

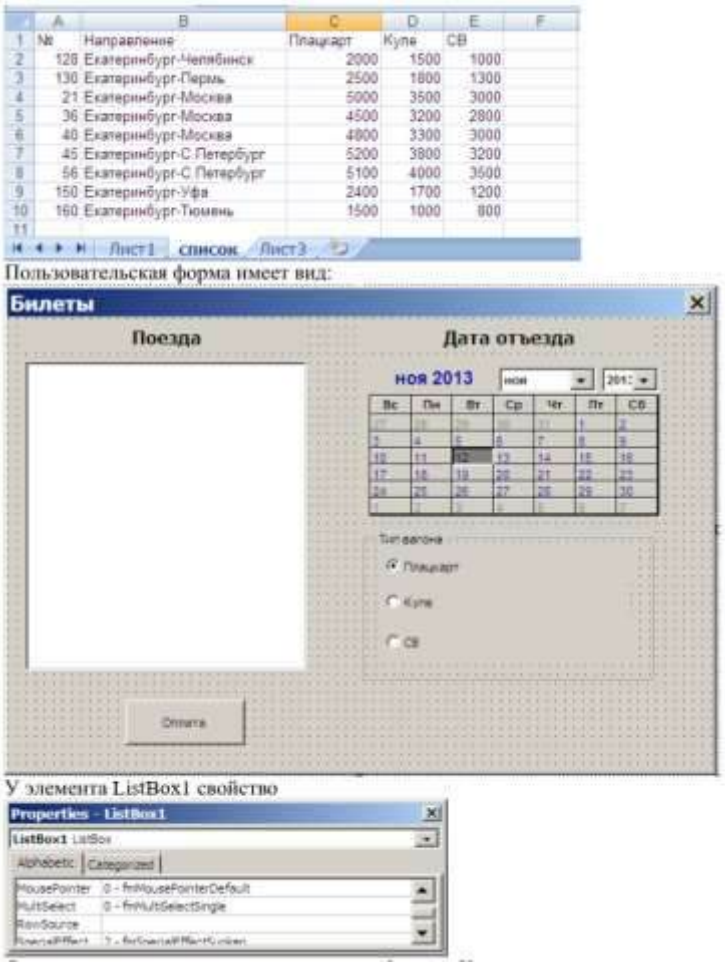


Рисунок 5 – Иллюстрация к заданию 1.2

Примечание. С примерами кодов программ можно ознакомиться по ссылке [7]

Лабораторная работа 2
Создание простейших игр на клеточном поле вVBA Excel (6)

Цель: научиться программировать игры на клеточном поле в VBA Excel

Задание 1. Запрограммировать игру «Жизнь», которая является клеточным автоматом.

Справочная информация

Игру "Жизнь" придумал британский математик Джон Конвей еще в 1970 году. Игра идет на большом (иногда даже бесконечном) поле в клеточку ("вселенной"). В один момент времени каждая клетка может быть в двух состояниях – живой (обозначим её единичкой) или же мертвой (пустой). Начальное состояние всех клеток в игре называют первым поколением.

Если брать блок клеток 3x3 с текущей клеткой в середине, то вокруг неё оказывается 8 клеток-соседей. Дальнейшая судьба клетки зависит от того, сколько именно живых клеток (N) окажется в этой окружающей области.

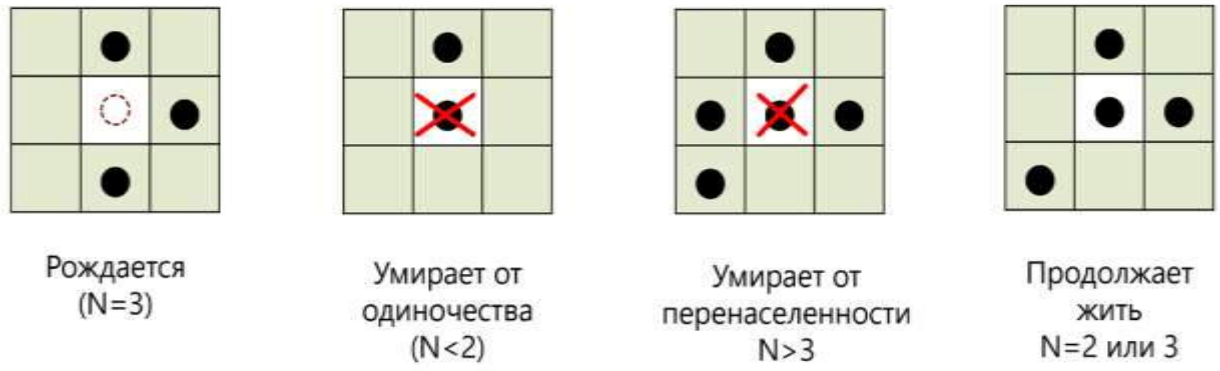


Рисунок 6 – Иллюстрация к заданию 2.1

Если клетка была пустая (мертвая), но у нее есть ровно 3 живых соседа, то в ней зарождается жизнь.

Если клетка живая, но у неё меньше 2 соседей, то она умирает от одиночества.

Если клетка живая, но у неё больше 3 соседей, то она умирает от перенаселения.

Если клетка живая и у нее 2-3 соседа, то клетка продолжает жить.

В математике подобные модели называют клеточными автоматами.

Задание 2. Запрограммировать игру «Крестики-нолики».

Задание 3. Запрограммировать игру «Пятнашки»

Примечание. С примерами кодов программ можно ознакомиться по ссылкам [8, 9]

Лабораторная работа 3
Создание приложений с использованием VBA Excel в Power Point (4)

Цель: научиться создавать приложения VBA Excel в Power Point

Задание 1. Создайте тест для проверки знания таблицы умножения с использованием VBA в Power Point.

Создайте программу, которая выполняет следующие действия:

• При нажатии кнопки «Новый пример» выводит случайным образом выбранные данные из диапазона от 0 до 9.

• После ввода тестируемым ответа и при нажатии кнопки «Проверить» выдает сообщение о правильности ответа, а в случае ошибочного ответа выдает не только сообщение об ошибке, но и правильный ответ.

• После каждого решенного примера выводит сообщения об общем количестве решенных примеров, решенных правильно и неправильно.

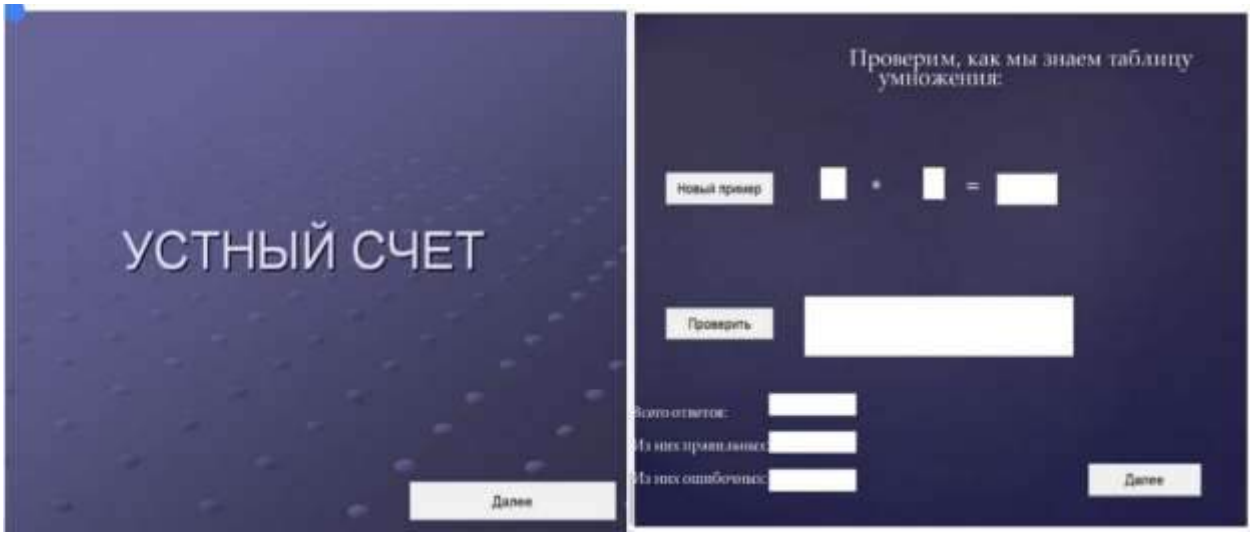


Рисунок 7 – Иллюстрация к заданию 3.1

Задание № 2. Создайте кроссворд с использованием VBA в Power Point

Примечание. С примерами кодов программ можно ознакомиться по ссылке [10]

СОДЕРЖАНИЕ КУРСА

Тема 1. Введение в визуальное программирование

Концепция визуального программирования. Графическое представление потока данных между компонентами системы. Технологии визуального описания алгоритмов. Концепция событийно-управляемого программирования.

Сферы применения визуального программирования. Преимущества и недостатки подходов визуального программирования.

Тема 2. Языки визуального программирования

Отличительные особенности языков визуального программирования: графические компоненты; формирование структурных связей; drag-and-drop редактор; многопоточность.

Классификации языков визуального программирования: учебные языки; языки, ориентированные на состояния; языки описания структуры данных; языки описания потока управления; языки описания потоков данных; языки описания процессов; языки для построения графического интерфейса.

Средства визуального программирования для разработки мобильных приложений и игр: классификация, назначение, преимущества и недостатки.

Тема 3. Визуальные среды разработки программ

Объектно-ориентированный подход программирования (ООП). Объект и классы. Атрибуты и методы. Принципы ООП – абстракция инкапсуляция, наследование, полиморфизм.

Реализация алгоритмических конструкций в визуальных средах программирования. Алгоритмические конструкции следование, условие, повторение. Циклы: циклы с известным числом повторений, циклы с предусловием и постусловием, безусловные циклы. Счетчики. Графические объекты. Координаты графических объектов. Клоны.

Концепция графических компонент в визуальном программировании. Работа с компонентами: свойства, события, методы. Основные свойства и методы ***формы***. Организация взаимодействия форм. Компоненты объединения элементов управления - контейнеры. Область прокрутки. Фреймы. Создание вида окна. Компоненты кнопки, их свойства, методы назначение. Простой список и комбинированный список Общая характеристика списков. Флажки и переключатели, их назначение и основные свойства.

Тема 4. Визуальные среды для разработки мобильных приложений и игр

Визуальная среда програмирования мобильного приложения. Классификация и виды мобильных приложений. Основные инструменты разработки мобильных приложений в визуальной среде програмирования.

Этапы проектирования мобильного приложения. Разработка интерфейса мобильного приложения. Основные компоненты мобильного приложения. Настройки экрана. Концепции UX/UI дизайна для мобильных приложений. Сенсоры мобильного устройства. Мультимедийные компоненты и хранилища. Использование онлайн-сервисов.

Проект игрового мобильного приложения для сферы социально-культурной деятельности. Концепция игрового приложение. Создание визуального решения мобильного приложения. Проектирование действия компонент приложения.

Программная реализация мобильного игрового приложения. Реализация форм интерфейса мобильного приложения. Программирование действий структурных компонент приложения. Тестирование приложения. Итоговая реализация мобильного приложения. Демонстрация и защита мобильного игрового проекта.

Тема 5. Визуальное программирование для создания цифровых 2D и 3D объектов сферы социально-культурной деятельности

Визуальный скриптинг. Системы графов и узлов в визуальном программировании. Система визуальных сценариев. Программирование потоков данных для управления рендерингом, создания шейдеров и текстур.

Платформы разработки графических продуктов с возможностями поддержки визуального скриптинга. Графические 3D редакторы. Платформы для создания анимации. Кросс-платформенные движки для создания видеоигр и моделирования.

Проект графического продукта для сферы социально-культурной деятельности. Концепция графического продукта. Выбор платформы графической реализации. Проектирование компонент, требующих программных решений. Конвертирование логики алгоритмов действия компонент в логику визуального скриптинга. Создание и тестирование визуального скриптинга. Итоговая реализация графического проекта. Демонстрация и защита графического продукта для сферы социально-культурной деятельности.

Тема 6. Разработка приложений в средах визуального програмирования

Интегрированные среды разработки программ. Создание проекта. Программирование с использованием классов. Реализация и наследование в приложениях с графическим интерфейсом.

Разработка приложений Windows в средах визуального программирования. Основные компоненты интерфейса пользователя. Использование расширенных элементов управления. Компоненты, обеспечивающие диалог с пользователем. Операции файлового ввода-выводы в Windows приложениях.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИЦИПЛИНЫ

Очная форма получения высшего образования

Разделы и темы	Количество аудиторных часов					Количество часов УСР	Форма контроля
	всего	лекции	лабораторные занятия	практические занятия	семинары		
Тема 1. Введение в визуальное программирование	2	2	2				
Тема 2. Языки визуального программирования	2	2					
Тема 3. Визуальные среды разрабки программ	2		2				
Тема 4. Визуальные среды для разработки мобильных приложений и игр	4		4			15	проект
Тема 5. Визуальное программирование для создания цифровых 2D и 3D объектов сферы социально-культурной деятельности	4		4			15	проект
Тема 6. Разработка приложений в средах визуального програмирования	16	2	14			26	проект
Всего аудиторных	**34**	**6**	**28**			**56**	**зачет**
Всего	**90**						

Заочная форма получения высшего образования

Разделы и темы	Количество аудиторных часов					Количество часов УСР	Форма контроля
	всего	лекции	лабораторные занятия	практические занятия	семинары		
Тема 1. Введение в визуальное программирование	1	1					
Тема 2. Языки визуального программирования	1	1					
Тема 3. Визуальные среды разрабки программ	1		1				
Тема 3. Визуальные среды для разработки мобильных приложений и игр	1		1			21	проект
Тема 4. Визуальное программирование для создания цифровых 2D и 3D объектов сферы социально-культурной деятельности	2		2			21	проект
Тема 5. Разработка приложений в средах визуального програмирования	2		2			40	проект
Всего аудиторных	**8**	**2**	**6**			**82**	**зачет**
Всего	**90**						

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

Основная литература

1. Тюгашев, А. А. Визуальное программирование: учебное пособие / А. А. Тюгашев. – Самара : СамГУПС, 2020. — ISBN 978-5-98941-325-6. — Текст : электронный // Лань : электронно-библиотечная система. – Режим доступа: URL: https://e.lanbook.com/book/161313 – Дата обращения: 08.04.2024. – С. 59. (Гриф – министерство транспорта РФ)

2. Хокинг, Джозеф. Unity в действии. Мультиплатформенная разработка на C# = Unity in Action. Multiplatform Game Development in C# / Джозеф Хокинг ; [пер. с англ. И. Рузмайкина]. - 2-е междунар. изд. - Санкт-Петербург : Питер, 2021. - 351 с.

3. Соколова, В.В. Вычислительная техника и информационные технологии. Разработка мобильных приложений : учебное пособие для вузов / В. В. Соколова. - Москва : Юрайт, 2020. – 175 с.

4. Савина, Н. В. Анимационная деятельность и сервис в туристической индустрии : учебное пособие для студентов учреждений высшего образования по специальностям "Экономика и управление туристической индустрией", "Туризм и природопользование", "Лингвистическое обеспечение международных коммуникаций (международный туризм)" / Н. В. Савина. - Минск : БГЭУ, 2020. – 350 с.

5. Трофимов, В. В. Алгоритмизация и программирование : учебник для студентов высших учебных заведений, обучающихся по экономическим направлениям / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. - Москва : Юрайт, 2020. – 136 с.

6. Булыгина, И. И. Анимация в сфере гостеприимства : учебник для направлений бакалавриата и магистратуры "Сервис", "Туризм", Гостиничное дело" / И. И. Булыгина, Е. Н. Гаранина, Н. И. Гаранин. - Москва : Кнорус, 2021. – 267 с.

Дополнительная литература

7. Ширева, С.Н. Практикум по VBA для Microsoft Excel: Учебное пособие. – РГППУ, 2017. – 126. – Точка доступа: https://elar.rsvpu.ru/bitstream/123456789/21894/1/Shireva_VBA.pdf. – Дата доступа: 23.05.2024.

8. Павлов, Н. Пишем игру "Жизнь" (Life) на VBA в Excel. – Планета Excel [Электронный ресурс]. – Точка доступа: https://www.planetaexcel.ru/techniques/3/10648/. – Дата доступа: 23.05.2024.

9. Логачёв, А. Пишем игры на... VBA. – DFT [Электронный ресурс]. – Точка доступа: https://dtf.ru/gamedev/71725-pishem-igry-na-vba-pt-1. – Дата доступа: 23.05.2024.

10. Петровская, Т. А. VBA программирование в PowerPoint [Электронный ресурс] : учебно-методическое пособие для студентов специальности 1-43 01 05 "Промышленная теплоэнергетика и теплотехника" / Т. А. Петровская, Е. И. Лозко, Д. Л. Кушнер ; Белорусский национальный технический университет, Кафедра "Промышленная теплоэнергетика и теплотехника". – Минск :БНТУ, 2013. – Точка доступа: https://rep.bntu.by/handle/data/5032. – Дата доступа: 23.05.2024.