МИНИСТЕРСТВО ОБРАЗОВАHИЯ И НАУКИ УКРАИHЫ

КАФЕДРА КОМПЬЮТЕРНЫХ ИНФОРМАЦИОННЫХ

ТЕХНОЛОГИЙ

Программирование Windows - приложенийКудий М.ИТ-00-2 8

КУРСОВАЯ РАБОТА ПО СИСТЕМНОМУ ПРОГРАММИРВАНИЮ

2006 г

**ЗАДАНИЕ**

**1. Индивидуальное задание:**

1. Разработать алгоритм расчета и реализовать его программно для решения следующей задачи:

Вычислить определенный интеграл 1) по формуле трапеций с тремя десятичными знаками.

Вычислить определенный интеграл 2) по формуле Симпсона при n=50

1) 2)

2. Изучить назначение и возможности компонентов средства разработки Delphi, их свойства, методы и доступные к обработке события, применить компоненты при разработке проекта: **GroupBox, SavePictureDialog**.

3. Создать DLL-библиотеку с использованием средства разработки Delphi, содержащую функцию, реализующую фрагмент алгоритма вычислений теоретической части индивидуального задания и возвращающую результат вычисления. Осуществить вызов процедуры (функции) из главного модуля приложения, при этом организовать загрузку подпрограмм из библиотеки путем динамической загрузки DLL-библиотеки.

4. Изучить назначение, организовать программно вызов и использование возвращаемых результатов следующими функциями Windows API. Для изучения функций использовать файл справки в стиле Windows Win32.hlp, находящийся в подкаталоге HELP каталога Delphi (используйте свои знания технического английского либо программы-переводчики типа Stylus), а также источник [2], номера страниц которого указаны после наименования функции:

ClipCursor 355, GetSystemInfo 103

5. Создать справочную подсистему, которая включает в себя:

1. справку в стиле Windows с обязательным включением следующих разделов:

- содержание тем справки;

- информация о предметной области приложения - определенных интегралах и методах их вычисления;

- инструкцию для неподготовленного пользователя по приемам работы с приложением.

При этом справка должна обеспечивать поиск по ключевым словам нужных тем и включать в себя пиктограммы (рисунки), являющиеся активными элементами для перехода к разделам справки;

2) Подсказки по различным элементам пользовательского интерфейса, появляющиеся при наведении на них курсора мыши.

3) Подменю в меню приложения, через команды которой вызываются общие сведения о приложении, инструкцию по работе со справкой, отдельные темы справки, информация о программе.

6. Разработать пользовательский интерфейс, включающий в себя:

1. - главную форму, меню приложения с подменю «Сервис» и командами, осуществляющими запись и открытие файлов, закрытие файлов и приложения и т.п., с подменю «Настройки» и командами, осуществляющими назначение режимов работы приложения, другими подменю;
2. - область отрисовки графического представления результатов работы приложения на главной форме;
3. - командные кнопки для выполнения ряда действий вашего приложения и элементы для ввода требуемой для этих действий информации (ввод данных должен производиться с проверкой их на соответствие ограничениям и выдачей соответствующих сообщений пользователю);
4. - диалоговое окно для назначения режимов работы вашего приложения при помощи переключателей;
5. - окна сообщений для информирования пользователя, подтверждения выполняемых действий, предупреждения об ошибках и их обработки.

7. Обработку ошибок ввода данных и открытия файлов (каталогов) провести при помощи:

защищенного блока try-finally-end и соответствующих фильтров обработки исключительных ситуаций.

8. Организовать при помощи соответствующих компонентов, объектов, их методов и свойств, для интеграла, чье значение вычислено по методу №1 значения нижней границы при фиксированных значениях верхней границы определенного интеграла; и обеспечить возможность сохранения изображения графика в файл.

**2. Объем и содержание курсовой работы:**

1. Использование современных средств визуального программирования под WINDOWS (Visual Basic, Delphi) для реализации разработанных алгоритмов.
2. Реализация концепции смешанного программирования в соответствии с индивидуальным заданием (использование библиотек DLL, функций WINDOWS API, объектных модулей на языке программирования Ассемблер и т.д.).
3. Реализация пользовательского интерфейса с обеспечением:

- эргономических требований;

- контекстно-зависимой помощи по управляющим элементам и отдельным формам приложения, включающей в себя подсказки, панель для отображения длинной части подсказки;

- структурированной по темам справки в стиле WINDOWS, содержащей руководство пользователю, с возможностью контекстного вызова соответствующих ее разделов;

- проверки вводимых пользователем данных на требуемые ограничения с указанием, каким должен быть правильный ввод;

- запросов на подтверждение выполнения действий, угрожающих целостности данных;

- возможности отказа от выполняемой обработки и отката выполненных операций на любом этапе.

1. Визуализация результатов обработки данных при помощи функций WINDOWS API с использованием контекстов устройств, режимов отображения, областей вывода и т.д., либо при помощи компонентов средств визуального программирования, в которых инкапсулированы вызовы соответствующих функций.
2. Вывод в файлы результатов обработки данных.
3. Обработка возможных ошибок (отсутствие файлов и каталогов, некорректные математические операции и т.д.) при помощи механизма обработки исключительных ситуаций в виде защищенных блоков программ, фильтров исключений и соответствующих объектов.
4. Соблюдение при программировании соглашений по наименованию переменных, констант и элементов управления (венгерской нотации), а также подробное документирование программных элементов путем использования комментариев.
5. Создание отлаженного под различные операционные системы и системное окружение приложения, что подразумевает возможность запуска и надежного функционирования приложения под управлением Windows 95 или Windows NT, при отсутствии специфических библиотек средств разработки, вне зависимости от структуры дерева каталогов и т.д.

**3. Пояснительная записка к курсовой работе должна содержать:**

1. - задание на проектирование;
2. - аннотацию курсовой работы;
3. - содержание;
4. - введение, в котором кратко излагается современное состояние операционных систем и средств разработки, а также обосновывается выбор того или иного средства разработки для реализации данной курсовой работы; - 2 с.
5. - описание предметной области проектирования, в котором кратко раскрывается содержание индивидуального задания, используемого для разработки алгоритмического обеспечения программной системы; - 2 с.
6. - описание процесса разработки программного комплекса (ПК), включающее в себя укрупненную структурно-функциональную схему ПК, описание содержания и функций модулей ПК с указанием используемых при этом форм и открываемых (создаваемых) файлов; - 3-4 с.
7. - эскиз элементов пользовательского интерфейса - меню, форм с перечнем управляющих элементов и указанием их назначения, области вывода графической информации с указанием содержания последней, структуры и краткого содержания справочной подсистемы (распечатка файла справки, используемого для компиляции справки в стиле WINDOWS, должна войти в приложения под названием «Руководство пользователю») ; - 4-5 с.
8. - методику использования в Вашей работе смешанного программирования: описания функций (процедур) DLL-библиотеки и способа ее построения и компиляции (листинг библиотеки должен войти в приложения), выданных в качестве индивидуального задания и использованных при программировании функций WINDOWS API (описание назначения функций и параметров их вызова); - 2 с.
9. - методику визуализации в Вашей работе результатов обработки данных (перечислить и дать краткое описание использованных компонентов, их свойств, методов и обработчиков событий); - 1-2 с.
10. - методику вывода в файлы результатов обработки данных (дать краткое описание технологии, объектов, свойств и методов, привести соответствующий фрагмент программы); - 1-2 с.
11. - требования к аппаратным и программным платформам, на которых может функционировать разработанное приложение; - 1 с.
12. - результаты самостоятельной работы, задания для которой выдаются индивидуально (описание функций Windows API и фрагменты листингов программы с их применением в вашем проекте; описание свойств и методов компонентов Delphi и фрагменты листингов программы с их применением в вашем проекте) – 3-4 с.;
13. - приложения, включающие: фрагменты документированных листингов, отражающих, по вашему мнению, ключевые моменты разработанных программных средств; руководство пользователю, в котором содержаться указания по правильной работе с разработанным программным средством и содержание справки в стиле Windows. - не более 5 с.

**АННОТАЦИЯ**

В работе рассмотрена программа вычисления значения и вывода графика определенного интеграла.

Работа содержит \_\_ страницы машинописного текста, \_\_ рисунков, \_\_ таблицы, \_\_ приложения.

Система Win32, ассемблер, Си, компилятор.

**СОДЕРЖАНИЕ**

ЗАДАНИЕ

АННОТАЦИЯ

ВВЕДЕНИЕ

1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ ПРОГРАММЫ

1.1 Численные методы вычисления определенных интегралов

1.1.1 Вычисление определенных интегралов методом трапеций

1.1.2 Вычисление определенных интегралов методом Симпсона

1.2 Описание среды программирования

2. СТРУКТУРА ПРОГРАММЫ

2.1 Описание процесса разработки

2.2 Функциональная схема программного комплекса

2.3 Реализация подынтегральных функций

2.4 Реализация функции разбора произвольно заданных математических функций

2.5 Реализация численных методов вычисления интегралов

2.5 Реализация функций построения и сохранения графиков

2.5.1 Основная концепция

2.5.2 Функция отрисовки графика

2.5.3 Описание методики сохранения графика в графический файл

2.6 Описание модулей и форм

2.7 Вызовы API-функций

2.8 Методика смешанного программирования

2.9 Контроль неквалифицированных действий пользователя

3. ЭЛЕМЕНТЫ ИНТЕРФЕЙСА ПРОГРАММНОГО КОМПЛЕКСА

3.1 Шаблоны окон

3.2 Структурно-функциональная схема программного комплекса

3.3 Описание компонентов: GroupBox, SavePictureDialog

4. ТЕСТОВЫЕ ВЫЗОВЫ ПРОГРАММЫ

5. ОПИСАНИЕ РАЗРАБОТКИ СПРАВОЧНОЙ СИСТЕМЫ

ЗАКЛЮЧЕНИЕ

ПЕРЕЧЕНЬ ССЫЛОК

ПРИЛОЖЕНИЕ А

ПРИЛОЖЕНИЕ Б

ПРИЛОЖЕНИЕ В

**ВВЕДЕНИЕ**

В настоящее время на рынке операционных систем наиболее широко распространенными являются операционные системы семейства Microsoft Windows. MS-Windows предоставляет пользователям оболочку графического интерфейса (GUI), которая обеспечивает стандартную среду пользователя и программиста. (GUI) предлагает более сложное и дружелюбное окружение пользователя, чем командно-управляемый интерфейс DOS. Работа в Windows основана на интуитивно понятных принципах. Вам легко переключиться с задачи на задачу и осуществлять обмен информацией между ними. Однако разработчики приложений традиционно сталкиваются с трудностями программирования, поскольку организация среды Windows является чрезвычайно сложной.

Delphi - язык и среда программирования, относящаяся к классу RAD- (Rapid Application Development ‑ «Средство быстрой разработки приложений») средств CASE - технологии. Delphi сделала разработку мощных приложений Windows быстрым процессом, доставляющим вам удовольствие. Приложения Windows, для создания которых требовалось большое количество человеческих усилий например в С++, могут быть написаны одним человеком, использующим Delphi.

Выгоды от проектирования в среде Windows с помощью Delphi:

- Устраняется необходимость в повторном вводе данных;

- Обеспечивается согласованность проекта и его реализации;

- Увеличивается производительность разработки и переносимость программ.

Визуальное программирование как бы добавляет новое измерение при создании создании приложений, давая возможность изображать эти объекты на экране монитора до выполнения самой программы. Без визуального программирования процесс отображения требует написания фрагмента кода, создающего и настрающего объект «по месту». Увидеть закодированные объекты было возможно только в ходе исполнения программы. При таком подходе достижение того, чтобы объекты выглядели и вели себя заданным образом, становится утомительным процессом, который требует неоднократных исправлений программного кода с последующей прогонкой программы и наблюдения за тем, что в итоге получилось.

Благодаря средствам визуальной разработки можно работать с объектами, держа их перед глазами и получая результаты практически сразу. Способность видеть объекты такими, какими они появляются в ходе исполнения программы, снимает необходимость проведения множества операций вручную, что характерно для работы в среде не обладающей визуальными средствами — вне зависимости от того, является она объектно-ориентированной или нет. После того, как объект помещен в форму среды визуального программирования, все его атрибуты сразу отображаются в виде кода, который соответствует объекту как единице, исполняемой в ходе работы программы.

Размещение объектов в Delphi связано с более тесными отношениями между объектами и реальным программным кодом. Объекты помещаются в вашу форму, при этом код, отвечающий объектам, автоматически записывается в исходный файл. Этот код компилируется, обеспечивая существенно более высокую производительность, чем визуальная среда, которая интерпретирует информацию лишь в ходе исполнения программы.

В мире многие разработчики твердо ориентируются на использование Delphi как на инструмент, позволяющий создавать высокоэффективные клиент-серверные приложения. Диапазон разработанных при помощи Delphi программных продуктов также поражает - от игровых программ до мощнейших банковских систем.

Таким образом, Delphi является одним из наиболее привлекательных для программиста средств разработки, качество создаваемых программ соответствуют уровню, обеспечиваемому компилятором языка С++. Все приведенные выше факторы обусловили выбор Delphi в качестве средства разработки программного комплекса в данной курсовой работе.

**1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ ПРОГРАММЫ**

**1.1 Численные методы вычисления определенных интегралов**

Задача вычисления интегралов возникает во многих областях прикладной математики. В большинстве случаев не удается найти аналитической формулы, т.е. выразить неопределенный интеграл в виде алгебраических и трансцендентных функций. Даже если аналитическая формула находится ,то она получается настолько сложной, что вычислять интеграл с ее помощью труднее, чем другими способами. Распространенными являются также случаи, когда подынтегральная функция задается графиком или таблицей экспериментально полученных значений. В таких ситуациях используют различные методы численного интегрирования, которые основаны на том, что интеграл представляется в виде предела интегральной суммы (суммы площадей), и позволяют определить эту сумму с приемлемой точностью.

**1.1.1 Вычисление определенных интегралов методом трапеций**

В данном методе заменяется на линейный интерполяционный многочлен, т.е. на элементарном отрезке [ *xi-1 , xi*] подынтегральная функция представляет собой отрезок прямой линии. Значение ***I*** в пределах [ *xi-1 , xi*] , равное площади криволинейной фигуры, заменяется площадью прямоугольной трапеции с высотой ***hi*** и основаниями ***f(xi-1) , f(xi)*** :

Si = 0.5 (yi-1 + yi) hi , i=1,2,...n .

После сложения этих соотношений получим формулу трапеций



I = 0.5 hi (yi-1 + yi) + R .

Если шаг интегрирования постоянный ( *hi = h = const* ) , то



I = h ( (y0 + yn)/2 + yi ) + R

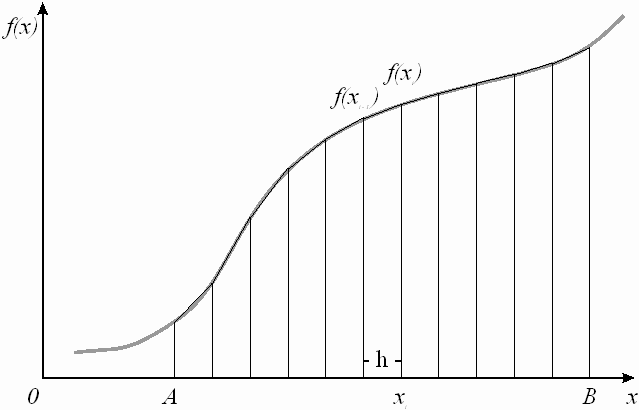


Рисунок 1 - Графическая интерпретация метода трапеций

**1.1.2 Вычисление определенных интегралов методом Симпсона**

Для вычисления интеграла



можно поступить таким способом.

Разобьем сначала промежуток [a, b] на некоторое число, n, равных промежутков

[x0, x1], [x1, x2], ..., [xn-1, xn] (x0 = a, xn = b),

искомый интеграл представится в виде суммы



После этого, к каждому из этих промежутков применим параболическое интерполирование, т.е. станем вычислять перечисленные интегралы по одной из приближенных формул: прямоугольников, трапеций или по параболической формуле.

В первых двух случаях мы получим уже известные формулы прямоугольников и трапеций.

Применим теперь параболическую формулу к каждому из интегралов, при этом положим, что



Получим



Складывая почленно эти равенства, получим формулу:

(8)



Эта формула называется формулой Симпсона. Ею пользуются для приближенного вычисления интегралов чаще, чем формулами прямоугольников и трапеций, так как она дает более точный результат.

**1.2 Описание среды разработки**

Delphi - это среда разработки, используемая прежде всего для создания и поддержки приложений, предназначенных как для отдельных персональных компьютеров, так и для серверов. Delphi, как и разработанные с ее помощью приложения, могут функционировать под практически любой 32 разрядной операционной системой типа Windows 95, 98, 2000, NT. Это довольно легкая в изучении среда, и в то же время довольно сложная. Изучить ее полностью и досконально – невозможно. Delphi имеет пользовательский графический интерфейс, подобный Visual Basic и C++. Человек, ранее работавший в подобной среде, не будет чувствовать себя не в своей тарелке. На данный момент множество фирм приняло за стандарт данный интерфейс для собственных приложений. Хорошим стимулом к получению знаний по данному предмету является знание хоть какого-нибудь языка программирования, или принципов написания программы. Идеально – знание языка программирования Pascal. Ведь весь исходный текст программы на Delphi пишется на языке Object Pascal, практически ничем не отличающимся от принципов, заложенных в такой знаменитой программной оболочке. Синтаксис, принцип модуля, процедуры, функции, все взято за основу.

О Delphi говорят как о среде быстрого создания приложений. Это технология визуального программирования, т.е. пользователь оформляет свою будущую программу, и видит результаты своей работы еще до запуска самой программы. В принципе, сам процесс написания приложения разделяется на две части. На первой стадии программист располагает на окна своей программы необходимые элементы, позиционирует, устанавливает нужные размеры, меняет свойства. На второй стадии – собственно, написание программного кода, описание свойств элементов, доступных только во время работы приложения, описание реакций на событие появление окна, нажатия на кнопку и др. Для задания каких-либо свойств элементу разрабатываемого приложения вовсе не обязательно писать массивные текстовые строки, достаточно изменить это свойство в инспекторе объектов (так называемом мониторе свойств выбранного элемента). Это изменение автоматически дополнит или модифицирует программный код.

Это большой плюс в визуальной технологии программирования. Создавая или модифицируя свой программный продукт, пользователь не зная или не обращая внимания на некоторые свойства элемента программы, а используя только необходимые, пишет полностью готовый рабочий продукт, порой выступающий на равных по сложности, с написанными на невизуальном редакторе.

Дополнительное удобство в работе в среде Delphi это мощная справочная система. Контекстно-зависимая от текущего выбранного элемента или строки программы, позволяет получить подробнейшую справку. Вложенные примеры позволяют, не отходя далеко от интересующей вас информации, просмотреть реализацию уже готовой, возможно полезной для вас, программы. Естественно, справка, как и сама среда разработки, описана на английском языке. Одновременно существуют и русифицированные файлы справки.

Контекстно-зависимое внедрение файлов справки позволяет поднять уровень проектируемого приложения на порядок выше. А не сказать про поддержку работы с базами данных – значит не сказать многого. При работе в среде программирования посредством так называемого BDE (Borland Database Engine), системного администратора баз данных, можно получать прямой доступ к таким стандартным форматам данных, как dBASE, Paradox, FoxPro, Access, ASCII таблицам. Набор драйверов Borland SQL Links обеспечивает все необходимые соединения с SQL-серверами.

Delphi использует структурный объектно-ориентированный язык (Object Pascal), который сочетает с одной стороны выразительную мощь и простоту программирования, характерную для языков 4GL, а с другой стороны эффективность языка 3GL. Программисты немедленно могут начать производить работающие приложения, и им не придется для этого изучать особенности программирования событий в Windows. Delphi полностью поддерживает передовые программные концепции включая инкапсуляцию, наследование, полиморфизм и управление событиями.

Редактирование программ можно осуществлять, используя запись и исполнение макросов, работу с текстовыми блоками, настраиваемые комбинации клавиш и цветовое выделение строк.

Delphi обладает мощнейшим, встроенным в редактор графическим отладчиком, позволяющим находить и устранять ошибки в коде. Вы можете установить точки останова, проверить и изменить переменные, при помощи пошагового выполнения в точности понять поведение программы. Если же требуются возможности более тонкой отладки, Вы можете использовать отдельно доступный Turbo Debugger, проверив ассемблерные инструкции и регистры процессора.

В состав RAD Pack входит эксперт для преобразования ресурсов, изготовленных в Borland Pascal 7.0, в формы Delphi. Уже появились эксперты, облегчающие построение DLL и даже написание собственных экспертов

**2. СТРУКТУРА ПРОГРАММЫ**

**2.1 Описание процесса разработки**

Процесс разработки программного комплекса реализующего задачи поставленные в задании к курсовому проекту начинается с создания нового проекта в среде программирования Delphi.

Главная форма программы содержит компоненты, необходимые для реализации пользовательского интерфейса, а также для выполнения иных необходимых функций:

- PageControl – компонент предназначен для создания многостраничного диалога: на одной странице будет расположен диалог расчета интегралов, а на второй – диалог построения графиков.

- MainMenu – главное меню программы.

- SavePictureDialog – диалог предназначенный для сохранения графических файлов (в программе используется для сохранения графика).

С помощью компонента PageControl реализуется многостраничный диалог. Первая страница диалога содержит компоненты:

- Panel – компонент используется в декоративных целях

- Label **-** служит для отображения текста на экране;

- ComboBox – служит для выбора информации из списка (используется для выбора подынтегральной функции)

- Edit – используется в программе для ввода значений (в программе используется для ввода численных значений)

- Button - позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы (в программе по нажатию кнопок происходит расчет интегралов).

Вторая страница диалога содержит компонент:

- Image – компонент используется для построения графика.

В программе необходимо реализовать следующие функции:

1. Реализация подынтегральных функций (реализовано в functions.dll).

2. Реализация функции разбора произвольно заданных математических функций (реализовано в модуле evalcomp.pas)

3. Реализация численных методов вычисления интегралов методами трапеций и Симпсона (реализовано в integrals.dll).

4. Реализация функции построения графика (реализовано в модуле Chart.pas).

5. Реализация функции сохранения графика в файл (реализовано в модуле Chart.pas).

6. Реализация вызовов API функций

Кроме того необходимо создать справочную систему помогающую пользователям в процессе работы с программой

**2.2 Функциональная схема программного комплекса**

Укрупненная функциональная схема приложения изображена на рис. 2. Схема показывает взаимодействие основных элементов программы.

Вход

Отрисовка главной формы

Главный цикл программы

Информация о программе

Справка

Меню программы

Вызов API-функций

Обработка произвольных функций

Вызов functions.dll

Вызов функций из integrals.dll и расчет интеграла

Отрисовка графиков функций

Выход

Динамическая загрузка в память dll-библиотек

Освобождение из памяти dll-библ.

Рисунок 2 – Функциональная схема приложения

Стрелки на рисунке указывают взаимодействие элементов программы и операционной системы. В виде прямоугольников на схеме изображены элементы программы и ОС.

**2.3 Реализация подынтегральных функций**

Подынтегральные функции, которые необходимы для вычисления интегралов  и, заданных в задании к курсовому проекту реализованы в отдельной динамической библиотеке functions.dll. Данная библиотека динамически присоединяется к программе в процессе работы. Листинги функций представлены на рисунке 3.

function Integral1Function(x: real; var y: real):boolean;

begin

Result:=true;

try

y:=1/sqrt(2+0.5\*x\*x)

except

result:=false;

end;

end;

function Integral2Function(x: real; var y: real):boolean;

begin

Result:=true;

try

y:=sin(2\*x)/sqr(x)

except

result:=false;

end;

end;

**2.4 Реализация функции разбора произвольно заданных математических функций**

В программе реализована возможность расчета интегралов и построения графиков не только для функций, заданных в процессе разработки программы, но и для математических функций задаваемых пользователем в процессе работы с программой. Данная функция реализована в модуле evalcomp.pas. Так как разработка данной возможности не была предусмотрена в задании к курсовому проекту, то реализации данного модуля не будет подробно рассматриваться в записке. Рассмотрим лишь правила использования функции.

Для использования данной возможности в главном модуле создается объект типа *evalvec*. Объект необходимо инициализировать строкой содержащей математическую функцию.

Пример:

new(calc, init('x'));

где calc – переменная типа *evalvec.*

Для вычисления значения заданной функции необходимо вызывать метод eval1d:

function eval1d(x:real):real;

В качестве параметра для данной функции необходимо задавать значение переменной х, на выходе получаем значение функции в точке x.

**2.5 Реализация численных методов вычисления интегралов**

Теоретические выкладки для вычисления интегралов численными методами были приведены выше в п.1.1.1 и п.1.1.2 данной записки. Программно данные методы реализованы в динамической бибилотеке integrals.dll. Листинги процедур для вычисления интегралов методами трапеций и Симпсона представлены на рисунках 4 и 5.

function Simpson(MainFunction: IntegralFunction;a,b:real;n: integer; var Res: real): boolean;

var

s, h, x,y: real;

i: integer;

begin

Result:=true;

try

s:=0;

h:=(b-a)/n;

for i:=0 to n do

begin

x:=a+h\*i;

if (i=0) or (i=n) then if MainFunction(x,y) then s:=s+y else Result:=false else

if i mod 2 = 0 then if MainFunction(x,y) then s:=s+2\*y else Result:=false else

if MainFunction(x,y) then s:=s+4\*y else Result:=false;

end;

except

Simpson:=false;

end;

Res:=s\*h/3;

end;

function CountTrap(MainFunction: IntegralFunction; a,b, eps: real; var Res: real): boolean;

function PTrap(MainFunction: IntegralFunction; a,b:real; DivNo: Integer; var Res: real): boolean;

var

x1, x2,y1,y2,y3,dx,sum: real;

i: integer;

begin

PTrap:=true;

try

sum:=0;

dx:=(b-a)/DivNo;

for i:=1 to DivNo-1 do

begin

x1:=a+dx\*i;

if MainFunction(x1,y1) then sum:=sum+y1 else Raise EAbort.Create('');

end;

sum:=sum\*2;

if MainFunction(a,y1) and MainFunction(b,y2) then

sum:=sum+y1+y2 else Raise EAbort.Create('');

Res:=sum\*dx/2;

except

PTrap:=False;

end;

end;

{-----}

var

T: integer;

i1, i2: real;

begin

countTrap:=true;

try

if PTrap(MainFunction,a,b,2,i1) and PTrap(MainFunction,a,b,4,i2) then

begin

T:=4;

while abs(i1-i2)>eps do

begin

if PTrap(MainFunction,a,b,t,i1) and PTrap(MainFunction,a,b,t\*2,i2) then t:=t\*2

else EAbort.Create('');

T:=T\*2;

end;

end else EAbort.Create('');

Res:=(i1+i2)/2;

except

CountTrap:=false;

end;

end;

**2.5 Реализация функций построения и сохранения графиков**

**2.5.1 Основная концепция**

В программе функции построения графика выделены в отдельный модуль chart.pas. Для реализации функций построения графика использованы принципы объектно-ориентированного программирования. Этот подход позволяет размещать на форме несколько областей для отображения графиков. Кроме того, на каждой области могут отображаться не один, а сразу несколько графиков различных функций. Для каждой области построения графиков необходимо создавать объекты класса TChart. При создании объекта в конструктор Create передается указатель на Canvas области построения графиков (в нашем случае в программе передается указатель на Canvas компонента TImage).

Для каждого графика, который необходимо отображать необходимо создавать объект класса TChartFunction. В качестве параметра в конструктор Create передается указатель на функцию, график которой необходимо отображать. После создания объекта класса TChartFunction необходимо вызывать метод TChart.AddFunction, который добавляет график текущей функции в список графиков объекта TChart.

**2.5.2 Функция отрисовки графика**

Метод DrawFuncitons класса TChart перебирает по очередности все графики из списка графиков и по очередности отрисовывает каждый из них используя для этого вызов процедуры TChart.DrawFunction. Листинг данной процедуры представлен на рис. 6.

procedure TChart.DrawFunction(funct: TChartFunction);

var

x, y: real;

x1,y1,x2,y2: integer;

a,b: boolean;

f: GraphFunction;

begin

x:=-(x0/FScale);

a:=false;

f:=funct.MainFunction;

while (x0+x\*FScale)<(Width-BorderRight) do

begin

if f(x,y) then

begin

x1:=round(x0+x\*FScale);

y1:=round(y0-y\*FScale);

if (x1>BorderLeft) and (x1<width-BorderRight) and (y1>BorderTop) and (y1<height-BorderBottom) then

begin

PutPixel(x1,y1, funct.Color);

end;

end;

x:=x+0.01;

end;

end;

В данной процедуре используя вызов функции MainFunction объекта класса TChartFunction определяются абсолютные координаты каждой точки графика в диапазоне значений, отображаемых на области отрисовки графика. Используя предварительно заданные масштаб и абсолютные координаты начала координат графика просчитываются координаты каждой точки графика.

**2.5.3 Описание методики сохранения графика в графический файл**

Для сохранения созданного графика в графический файл используется метод SaveToFile класса TChart. В качестве параметра в данный метод передается имя файла в который необходимо сохранять изображение.

Данная процедура создает временный объект класса TPicture, копирует канвас графика в канвас вновь созданного объекта и после этого, используя метод TPicture.SaveToFile сохраняет изображение в файл. Полный листинг данного метода представлен на рисунке 7.

procedure TChart.SaveToFile(filename: string);

var

temporary: TPicture;

begin

temporary:=TPicture.Create;

temporary.Bitmap.Width:=width;

temporary.Bitmap.Height:=height;

Temporary.Bitmap.Canvas.CopyRect(Temporary.Bitmap.Canvas.ClipRect,MainCanvas, MainCanvas.ClipRect);

Temporary.SaveToFile(filename);

temporary.Destroy;

end;

**2.6 Описание модулей и форм**

MainForm (MainUnit.pas) – главная форма программы на которой отображаются вычисления интегралов, а также происходит построение графиков функций

AboutForm (Abut.unit) – форма содержащая сведения о разработчике программы

EvalForm (EvalFormUnit.pas) – форма для ввода произвольных математических функций

SystemInfoForm (SysInfo.pas) – форма отображающая результат выполнения API-функции GetSystemInfo

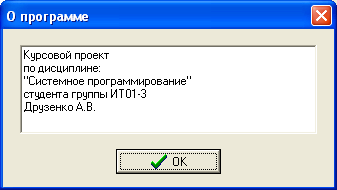
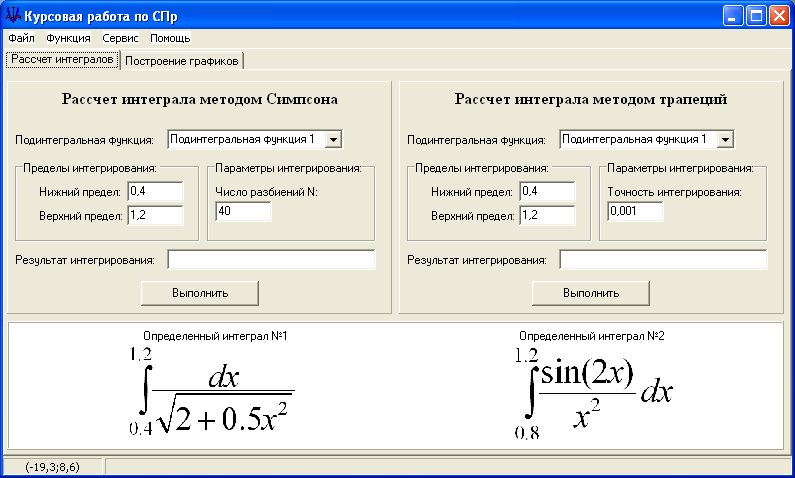
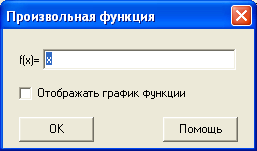
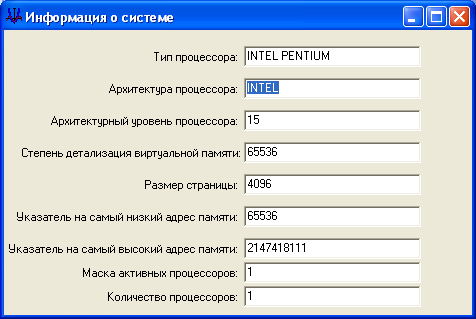


Рисунок 8 – Схема взаимодействия форм приложения

**2.7 Вызовы API-функций**

В качестве задания к курсовому проекту необходимо описать и использовать в программе две API-функции: ClipCursor и GetSystemInfo.

1. Функция ClipCursor содержится в стандартной библиотеке user32.dll

Описание функции:

function ClipCursor(Rect: TRect);

Эта функция заключает курсор в Rect. Если Rect имеет значение nil, то курсор является неограниченным.

Параметры:

Rect: ограничивающий TRect в координатах экрана.

Пример использования данной функции в программе приведен в листинге на рисунке 9.

procedure TMainForm.ClipCursor1Click(Sender: TObject);

var

R: TRect;

begin

ClipCursor1.Checked:=not ClipCursor1.Checked;

if ClipCursor1.Checked then

begin GetClipCursor(ClipRect);

Left:=MainForm.Left;

Right:=MainForm.Left+MainForm.Width;

Bottom:=MainForm.Top+MainForm.Height;

end;

ClipCursor(@R);

end else ClipCursor(@ClipRect);

end;

2. Функция GetSystemInfo

Описание функции:

procedure GetSystemInfo(lpSystemInfo: TSystemInfo);

Функция в качестве параметра получает указатель на структуру SystemInfo, которую она заполняет. Структуру SystemInfo содержит информацию о текущей системе.

dwOemId: DWord

Устаревший элемент, предназначенный для совместимости с предыдущими версиями Windows NT (3.5 и ранее). Начиная с Windows 3.51 приложения должны использовать переход wProcessorArchitecture объединения. Windows 95/98/Me: система всегда устанавливает этот элемент, чтобы обнулить значение, определенное для PROCESSOR\_ARCHITECTURE\_INTEL.

wProcessorArchitecture:Word

Определяет архитектуру процессора системы. Этим элементом может быть одно из следующих значений:

PROCESSOR\_ARCHITECTURE\_UNKNOWN

PROCESSOR\_ARCHITECTURE\_INTEL

PROCESSOR\_ARCHITECTURE\_MIPS - Windows NT 3.51

PROCESSOR\_ARCHITECTURE\_ALPHA - Windows NT 4.0 и ранее

PROCESSOR\_ARCHITECTURE\_PPC - Windows NT 4.0 и ранее

PROCESSOR\_ARCHITECTURE\_IA64 - 64-битная версия Windows

PROCESSOR\_ARCHITECTURE\_IA32\_ON\_WIN64 - 64-битная версия

Windows PROCESSOR\_ARCHITECTURE\_AMD64 - 64-битная версия

Зарезервирован для будущего использования.

dwPageSize : DWord

Определяет размер страницы и степень детализации защиты страницы и блокирования. Это - размер страницы, используемый функцией VirtualAlloc.

lpMinimumApplicationAddress: Pointer

Указатель на самый низкий адрес памяти, доступный для приложений и библиотек динамической связи (DLLs).

lpMaximumApplicationAddress: Pointer

Указатель на самый высокий адрес памяти, доступный для приложений и DLLs.

dwActiveProcessorMask: DWord

Определяет маску, представляющую набор процессоров, конфигурированных в систему. Бит 0 - процессор 0; бит 31 - процессор 31.

dwNumberOfProcessors: DWord

Определяет количество процессоров в системе.

dwProcessorType: DWord

Устаревший элемент, предназначенный для совместимости с предыдущими версиями Windows NT (3.5 и ранее). Windows 95/98/Me: Определяет тип процессора в системе. Windows NT: Этот элемент больше не имеет значения, но сохранен для совместимости с Windows 95 и предыдущими версиями Windows NT. Необходимо использовать wProcessorArchitecture, wProcessorLevel, и wProcessorRevision элементы, чтобы определить тип процессора. Этим элементом может быть одно из следующих значений: PROCESSOR\_INTEL\_386 PROCESSOR\_INTEL\_486 PROCESSOR\_INTEL\_PENTIUM PROCESSOR\_MIPS\_R4000 - Windows NT PROCESSOR\_ALPHA\_21064 - Windows NT

dwAllocationGranularity: DWord

Определяет степень детализации распределения виртуальной памяти.

wProcessorLevel : Word

Windows 95 - этот элемент не поддерживается. Windows NT - определяет архитектурный уровень процессора.

**2.8 Методика смешанного программирования**

При запуске нескольких экземпляров одного приложения, Windows загружает в оперативную память только одну копию кода и ресурсов - модуль приложения, создавая несколько отдельных сегментов данных, стека и очереди сообщений, каждый набор которых представляет из себя задачу, в понимании Windows. Копия приложения представляет из себя контекст, в котором выполняется модуль приложения. DLL - библиотека также является модулем. Она находится в памяти в единственном экземпляре и содержит сегмент кода и ресурсы, а также сегмент данных. DLL - библиотека, в отличие от приложения не имеет ни стека, ни очереди сообщений. Функции, помещенные в DLL, выполняются в контексте вызвавшего приложения, пользуясь его стеком. Но эти же функции используют сегмент данных, принадлежащий библиотеке, а не копии приложения.

В силу такой организации DLL, экономия памяти достигается за счет того, что все запущенные приложения используют один модуль DLL, не включая те или иные стандартные функции в состав своих модулей.

Часто, в виде DLL создаются отдельные наборы функций, объединенные по тем или иным логическим признакам, аналогично тому, как концептуально происходит планирование модулей ( в смысле unit ) в Pascal. Отличие заключается в том, что функции из модулей Pascal компонуются статически - на этапе линковки, а функции из DLL компонуются динамически, то есть в run-time.

Таким образом, DLL позволяют:

* избежать громоздкости исполняемого кода прикладной программы;
* улучшить структурированность и модульность приложения;
* уменьшить занимаемый приложением объем памяти за счет того, что несколько одновременно выполняющихся программ могут использовать одну и ту же копию подпрограммы из DLL, при этом в памяти будет физически загружена только одна копия библиотеки;
* использовать любую библиотеку в любых программах, не зависимо от того, на каком языке была написана программа и библиотека.

DLL являются одними из наиболее важных ключевых элементов при написании любого приложения Windows. Само ядро Windows представлено тремя большими по объему динамическими библиотеками: KERNEL.DLL, USER.DLL и GDI.DLL. Файл KERNEL.DLL, например, отвечает за управление памятью, процессами и потоками; USER.DLL содержит функции интерфейса пользователя, необходимые для создания окон и обработки сообщений Windows; на GDI.DLL возложена работа с графикой. Общее количество используемых самой Windows динамических библиотек составляет порядка двух тысяч.

В курсовом проекте также используется методика смешанного программирования. Так подынтегральные функции, используемые для расчета интегралов вынесены в отдельную динамическую библиотеку functions.dll. Численные методы вынесены также в отдельную библиотеку integrals.dll.

В основной программе используется динамическая загрузка dll-библиотек. Весь процесс загрузки библиотек вынесен в отдельную процедуру, листинг которой приведен на рисунке 10.

procedure TMainForm.LoadDlls;

var

x: real;

begin

IntegralsDll:=LoadLibrary('Integrals.dll');

FunctionsDll:=LoadLibrary('Functions.dll');

if IntegralsDll=0 then

begin

Application.MessageBox('Не найден модуль ''Integrals.dll''.

Работа приложения будет прекращена',

'Ошибка!!!', MB\_Ok or MB\_ICONERROR);

Application.Terminate;

end;

if FunctionsDll=0 then

begin

Application.MessageBox('Не найден модуль ''Functions.dll''. Работа приложения будет прекращена',

'Ошибка!!!', MB\_Ok or MB\_ICONERROR);

Application.Terminate;

end;

@Simpson:=GetProcAddress(IntegralsDLL,'Simpson');

@CountTrap:=GetProcAddress(IntegralsDLL,'CountTrap');

if (@CountTrap=nil) or (@Simpson=nil) then

begin

Application.MessageBox('Ошибка в модуле ''Integrals.dll''.

Работа приложения будет прекращена',

'Ошибка!!!', MB\_Ok or MB\_ICONERROR);

Application.Terminate;

end;

@Integral1Function:=GetProcAddress(FunctionsDLL,'Integral1Function');

@Integral2Function:=GetProcAddress(FunctionsDLL,'Integral2Function');

if (@Integral1Function=nil) or (@Integral1Function=nil) then

begin

Application.MessageBox('Ошибка в модуле ''Functions.dll''.

Работа приложения будет прекращена',

'Ощибка!!!', MB\_Ok or MB\_ICONERROR);

end;

**2.9 Контроль неквалифицированных действий пользователя**

В приложении предусмотрен контроль за неквалифицированными действиями пользователя. Программа разработана таким образом, чтобы при любых действиях пользователя не возникало сбоя в программе.

Выполнение данной задачи достигалось путем использования структурной обработки исключительных ситуаций- системы, позволяющей программисту при возникновении ошибки (исключительной ситуации) связаться с кодом программы, подготовленным для обработки такой ошибки. Это выполняется с помощью языковых конструкций, которые как бы “охраняют” фрагмент кода программы и определяют обработчики ошибок, которые будут вызываться, если что-то пойдет не так в “охраняемом” участке кода. В данном случае понятие исключительной ситуации относится к языку и не нужно его путать с системными исключительными ситуациями (hardware exceptions), такими как General Protection Fault. Исключительные ситуации в Delphi же независимы от “железа”, не используют прерываний и используются для обработки ошибочных состояний, с которыми подпрограмма не готова иметь дело. Системные исключительные ситуации, конечно, могут быть перехвачены и преобразованы в языковые исключительные ситуации, но это только одно из применений языковых исключительных ситуаций.

При традиционной обработке ошибок, ошибки, обнаруженные в процедуре обычно передаются наружу (в вызывавшую процедуру) в виде возвращаемого значения функции, параметров или глобальных переменных (флажков). Каждая вызывающая процедура должна проверять результат вызова на наличие ошибки и выполнять соответствующие действия. Часто, это просто выход еще выше, в более верхнюю вызывающую процедуру и т.д. : функция A вызывает B, B вызывает C, C обнаруживает ошибку и возвращает код ошибки в B, B проверяет возвращаемый код, видит, что возникла ошибка и возвращает код ошибки в A, A проверяет возвращаемый код и выдает сообщение об ошибке либо решает сделать что-нибудь еще, раз первая попытка не удалась.

Структурная обработка исключительной ситуации замещает ручную обработку ошибок автоматической, сгенерированной компилятором системой уведомления. В приведенном выше примере, процедура A установила бы “охрану” со связанным обработчиком ошибки на фрагмент кода, в котором вызывается B. B просто вызывает C. Когда C обнаруживает ошибку, то создает (*raise*) исключительную ситуацию. Специальный код, сгенерированный компилятором и встроенный в Run-Time Library (RTL) начинает поиск обработчика данной исключительной ситуации. При поиске “защищенного” участка кода используется информация, сохраненная в стеке. В процедурах C и B нет такого участка, а в A - есть. Если один из обработчиков ошибок, которые используются в A, подходит по типу для возникшей в C исключительной ситуации, то программа переходит на его выполнение. При этом, область стека, используемая в B и C, очищается; выполнение этих процедур прекращается.

Если в A нет подходящего обработчика, то поиск продолжается в более верхнем уровне, и так может идти, пока поиск не достигнет подходящего обработчика ошибок среди используемых по умолчанию обработчиков в RTL. Обработчики ошибок из RTL только показывают сообщение об ошибке и форсированно прекращают выполнение программы. Любая исключительная ситуация, которая осталась необработанной, приведет к прекращению выполнения приложения.

Без проверки возвращаемого кода после каждого вызова подпрограммы, код программы должен быть более простым, а скомпилированный код - более быстрым. При наличии исключительных ситуаций подпрограмма B не должна содержать дополнительный код для проверки возвращаемого результата и передачи его в A. B ничего не должна делать для передачи исключительной ситуации, возникшей в C, в процедуру A - встроенная система обработки исключительных ситуаций делает всю работу.

Данная система называется *структурной,* поскольку обработка ошибок определяется областью “защищенного” кода; такие области могут быть вложенными. Выполнение программы не может перейти на произвольный участок кода; выполнение программы может перейти только на обработчик исключительной ситуации активной программы.

Пример использования структурной обработки исключительных ситуаций с использованием блока try-except в процедуре TMainForm.btnSimpsonRunClick приведен на рисунке 11.

try

a:=StrToFloat(edSimpA.Text);

b:=StrToFloat(edSimpB.Text);

n:=StrToInt(edSimpN.Text);

if a>b then

begin

Application.MessageBox('Нижний предел больше верхнего', 'Ошибка!',MB\_OK or MB\_ICONError);

exit;

end;

except

on EConvertError do

begin

Application.MessageBox('Ошибка ввода численных значений', 'Ошибка!',MB\_OK or MB\_ICONError);

exit;

end;

end;

Также в программе применяется методика создания и обработки пользовательской исключительной ситуации. Так в модуле evalComp.pas создается класс EvalException, являющийся потомком класса EAbort. В процессе работы с модулем может произойти генерация исключительной ситуации, которую в свою очередь будет обрабатывать вызывающий модуль (в нашем случае процедура TEvalForm.btnOkClick). Листинг примера работы с пользовательскими исключительными ситуациями приведен на рисунке 12.

type

EvalException = class(EAbort)

end;

function myfac(x,y:real):real;

var

begin

if x<0 then begin raise EvalException.Create(''); halt; end;

if x = 0 then myfac := 1

else

end;

procedure TEvalForm.btnOkClick(Sender: TObject);

begin

Dispose(calc, done);

try

new(calc, init(edEval.Text)); Close;

except

on EvalException do ShowMessage('Ошибка в формуле');

end;

end;

Кроме того, в программе для предотвращения возникновения непредвиденных ситуаций применяется защищенный блок try-finally-end. Листинг приведен на рисунке 13.

procedure TChart.SaveToFile(filename: string);

var temporary: TPicture;

begin

temporary:=TPicture.Create;

try

temporary.Bitmap.Width:=width; temporary.Bitmap.Height:=height;

Temporary.Bitmap.Canvas.CopyRect(Temporary.Bitmap.Canvas.ClipRect,MainCanvas, MainCanvas.ClipRect);

Temporary.SaveToFile(filename);

finally

temporary.Destroy;

end;

end;

**3. ЭЛЕМЕНТЫ ИНТЕРФЕЙСА ПРОГРАММНОГО КОМПЛЕКСА**

* 1. **Шаблоны окон**

На рисунках 12 и 13 изображено главное окно программы, которое может изменять свой вид в зависимости от выбранной закладки. На рисунке 14 изображено окно, предназначенное для ввода произвольной функции. На рисунке 15 представлена форма окна отображающего сведения о системе. Окно на рисунке 16 изображено окно сведений о разработчике.

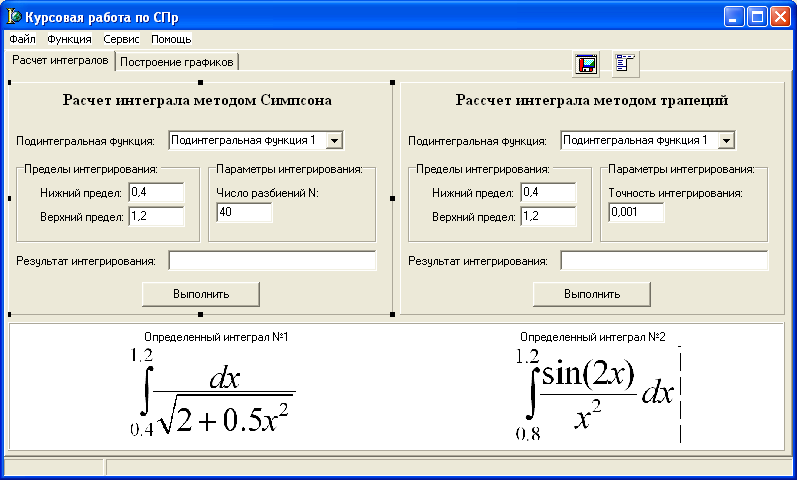


Рисунок 12 – Главное окно программы – закладка «Расчет интегралов»

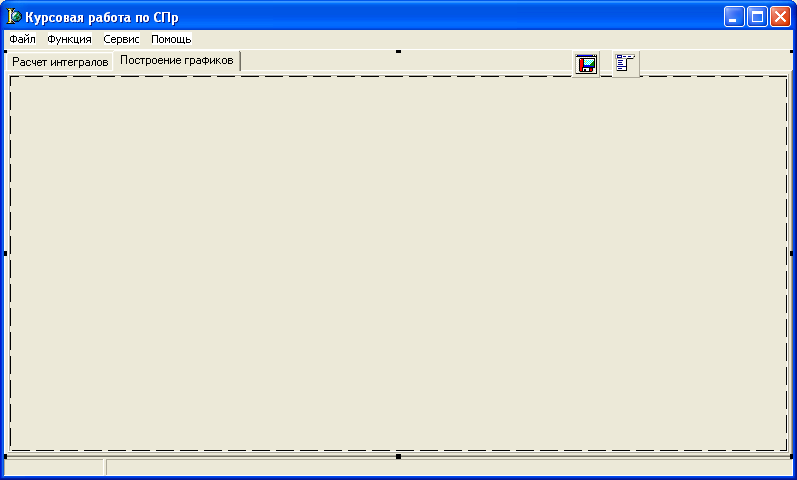


Рисунок 13 – Главное окно программы – закладка «Построение графика»

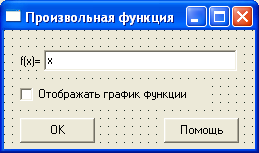


Рисунок 14 – Окно «Произвольная функция»

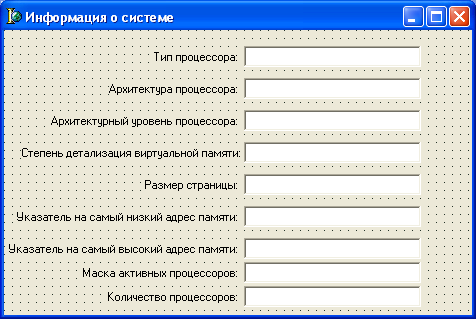


Рисунок 15 – Окно отображающее информацию о текущей системе

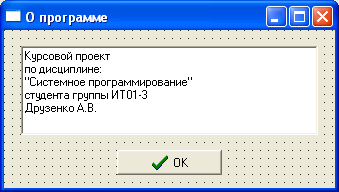


Рисунок 16 – Окно сведений о программе

Все компоненты, которые расположены на формах представленных на рисунках 12-16, описаны в таблицах 1-4.

**3.2 Структурно-функциональная схема программного комплекса**

Файл формы – MainUnit.pas

Имя формы – MainForm

Заголовок – Курсовая работа по СПр

Назначение формы – ввод исходных данных для вычисления интегралов, для отрисовки графика зависимости значения интеграла от значения верхнего предела интегрирования при фиксированном нижнем пределе, вывод на экран полученного графика, полученных результатов расчета интегралов и вызов дополнительных форм, демонстрирующих работу API-функций, формы для ввода произвольной функции, формы для получения информации о разработчике, а также вызова справки в стиле Windows.

В таблице 1 представлен перечень компонентов содержащихся на форме.

Таблица 1 – Описание главной формы приложения

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Имя компонента в модуле** | | | **Назначение компонента** | | | **События компонента** | | | **Назначение обработчиков событий** | **Примечания** | | |
| cbSimpFunct, cbTrapFunct | | | Компонент ComboBox предназначенный для выбора подинтегральной функции | | | - | | | - | - | | |
| edSimpA, edSimpB, edTrapA, edTrapB | | | Поля для ввода изменения границ интегрирования | | | - | | | - | - | | |
| edSimpN | | | Поле для ввода числа разбиений в методе Симпсона | | | - | | | - | - | | |
| edTrapEPS | | | Поля для ввода точности вычислений в методе трапеций | | | - | | | - | - | | |
| edSimpResult,  edTrapResult | | | Поля для вывода результатов интегрирования | | | - | | | - | - | | |
| btnSimpsonRun | | | Кнопка для расчета интеграла методом Симпсона | | | onClick | | | Сначала производиться проверка на наличие верхней и нижней границ. Далее проверка на то, чтобы нижняя граница была меньше или равна верхней. Контроль записи вещественных чисел. В случае прохождения всех вышеперечисленных проверок—расчет методом Симпсона и  выдача в поле вывода результата. Вывод значения интеграла в edSimpResult. | MessageBox('Нижний предел больше верхнего', 'Ошибка!',MB\_OK or MB\_ICONError);  MessageBox('Ошибка ввода численных значений', 'Ошибка!',MB\_OK or MB\_ICONError) | | |
| btnTrapRun | | | Кнопка для расчета интеграла методом трапеций | | | onClick | | | Сначала производиться проверка на наличие верхней и нижней границ, проверка на то, чтобы нижняя граница была меньше или равна верхней. Контроль записи вещественных чисел.—расчет методом трапеций и  выдача в поле вывода результата. Вывод значения интеграла в TrapResult. | MessageBox('Нижний предел больше верхнего', 'Ошибка!',MB\_OK or MB\_ICONError);  MessageBox('Ошибка ввода численных значений', 'Ошибка!',MB\_OK or MB\_ICONError) | | |
| imInt1, imInt2 | | Image для отображения интегралов 1 и 2 | | | - | | - | | | | - | |
| PageControl | | Компонент TPageControl, для управления закладками | | | - | | - | | | | - | |
| ChartBox | | Компонент Image для отрисовки графиков | | | OnMouseDown | | Выполняется проверка какая клавиша нажата – левая или правая – устанавливается соответствующий флаг - и в результате принимается решение о начале масштабирования либо перемещения графика | | | | - | |
| OnMouseMove | | Выполняется проверка флага нажатия кнопки мыши и в результате: если нажата левая клавиша – продолжение перемещения, правая – продолжение масштабирования | | | | - | |
| OnMouseUp | | Выполняется проверка флага нажатия кнопки мыши и в результате: если нажата левая клавиша – окончание перемещения и снятие флага, правая – окончание масштабирования и снятие флага | | | | - | |
| SavePictureDialog | | Компонент страницы DIALOGS служащий для сохранения изображения графика в виде отдельного файла. | | | Execute | | Вызов стандартного окна сохранения файлов | | | |  | |
| MainMenu | | Главное меню программы состоящее из разделов: «Файл», «Фукции», «Сервис», «Помощь» | | | - | | - | | | | - | |
| mmSave | | Пункт меню для сохранения графика | | | onClick | | Вызов стандартного окна сохранения графика | | | |  | |
| mmExit | | Пункт меню для выхода из программы | | | onClick | | Выход программы | | | |  | |
| mmIntFunct1 | | Пункт меню для отображения/скрытия графика подынтегральной функции 1 | | | onClick | | Отображает/скрывает график подынтегральной функции 1 | | | |  | |
| mmIntFunct2 | | Пункт меню для отображения/скрытия графика подынтегральной функции 2 | | | onClick | | Отображает/скрывает график подынтегральной функции 2 | | | |  | |
| mmIntegral1 | | Пункт меню для отображения/скрытия графика интеграла 1 | | | onClick | | Отображает/скрывает графикинтеграла 1 | | | |  | |
| mmEval | | Пункт меню для открытия окна ввода произвольной функции | | | onClick | | Вызывает окно ввода произвольной функции (форма EvalForm) | | | |  | |
| mmGrid | | Пункт меню для установки сетки на графике | | | onClick | | Разрешает/запрещает отрисовку сетки на графике | | | |  | |
| mmSysInfo | Пункт меню для вывода информации о системе | | | onClick | | | | Выводит окно с информацией о системе (форма SystemInfoForm) | | | |  |
| mmClipCursor | Пункт меню для использования функции ClipCursor | | | onClick | | | | Демонстрирует использование функции ClipCursor | | | |  |
| mmHelp | Пункт меню для вызова помощь | | | onClick | | | |  | | | |  |
| mmAbout | Пункт меню для вывода окна информации о разработчике | | | onClick | | | | Вызов формы FormAbout с информацией о разработчике | | | |  |

Файл формы – evalForm.pas

Имя формы – EvalForm

Заголовок – Произвольная функция

Описание – форма предназначена для ввода произвольной пользовательской функции, отображении/скрытия ее графика.

В таблице 2 представлен перечень компонентов содержащихся на форме.

Таблица 2 – Описание формы ввода произвольной функции

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя компонента в модуле** | **Назначение компонента** | **События компонента** | **Назначение обработчиков событий** | **Примечания** |
| edEval | Поле для ввода произвольной функции | - | - | - |
| cbEval | CheckBox для отображения/скрытия графика произвольной функции | - | - | - |
| btnOk | Функция принятия изменений в произвольной функции | onClick | Выполняется проверка введенной функции, если функция соответствует правилам, то происходит изменение произвольной функции, иначе выводится сообщение | ShowMessage('Ошибка в формуле'); |

Файл формы – SysInfo.pas

Имя формы – SystemInfoForm

Заголовок - Информация о системе

Описание – форма предназначена для вывода информации о текущей системе

Таблица 3 – Описание формы информации о системе

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя компонента в модуле** | **Назначение компонента** | **События компонента** | **Назначение обработчиков событий** |
| SystemInfoForm | Форма для отображения информации о системе | onCreate | Вызывается функция GetSystemInfo и результат заносится в поля формы |
| leProcessorType, leProcessorArchitecture, leProcessorLevel,  leAllocationGranularity, lePageSize, leMinimumApplicationAddress, leMaximumApplicationAddress, leActicveProcessorMask, leNumberOfProcessor | Компоненты для отображения информации о системе |  |  |

Файл формы – About.pas

Имя формы – AboutForm

Заголовок - О программе

Описание – форма предназначена для вывода информации о разработчике

Таблица 4 – Описание формы информации о разработчике

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя компонента в модуле** | **Назначение компонента** | **События компонента** | **Назначение обработчиков событий** | **Примечания** |
| meAabout | Поле для вывода информации о разработчике | - | - | - |
| btnOk | Кнопка закрытия формы | onClick | Закрытие формы |  |

**3.3 Описание компонентов: GroupBox, SavePictureDialog**

1. Компонент TGroupBox

Модуль - StdCtrls

Описание:

TGroupBox используется для оформления внешнего вида приложения и для группировки установленных в форму компонентов по смыслу. Все компоненты в форме можно разделить по функциональным назначениям и разместить в таких компонентах. Удобен при назначении порядка перехода по компонентам с помощью кнопки Tab. Заголовок этого компонента изменяется свойством **Caption.**

Приведем иерархию классов для компонента TGroupBox

TObject -> TPersistent -> TComponent -> TWinControl -> TCustomControl -> TCustomGroupBox -> TGroupBox

Основные свойства, методы и события для этого компонента приведены в таблицах 5,6 и 7.

Таблица 5 – Основные свойства компонента ТGroupBox

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| property Components[Index: Integer]: TComponent; | Список всех компонентов, для которых данный компонент является владельцем. |
| property Brush: TBrush; | Определяет цвет и шаблон для отрисовки фона копмонента. |
| property ClientOrigin: TPoint; | Возвращает координаты экрана в пикселях левого верхненго угла клиентской области компонента. |
| property TabOrder: TTabOrder; | Отображает номер в последовательности при переходе по компонента с помощью |
| property Controls[Index: Integer]: TControl; | Список всех дочерних компонентов |

Таблица 6 – Основные методы компонента ТGroupBox

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| procedure Broadcast(var Message); | Посылает сообщение каждому из дочерних компонентов |
| function CanFocus: Boolean; dynamic; | Возвращает значение, может ли компонент принимать фокус. |
| Procedure InsertControl(AControl: TControl); | Добавляет компонент в список дочерних компонентов данного |
| function FindChildControl(const ControlName: string): TControl; | Возвращает указатель на дочерний компонент по его имени. |
| procedure SetFocus; virtual; | Устанавливает фокус на компонент |

Таблица 7 – Основные события компонента ТGroupBox

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| property OnEnter: TNotifyEvent; | Событие вызывается когда компонент получает фокус |
| property OnExit: TNotifyEvent; | Событие вызывается когда компонент теряет фокус |
| property OnClick: TNotifyEvent; | Событие вызывается когда происходит клик на компоненте |
| property OnDblClick: TNotifyEvent; | Событие вызывается при двойном нажатии левой кнопки мыши на компоненте |
| property OnMouseDown: TMouseEvent; | Событие вызывается когда происходит нажатие кнопки мыши при установленном на компоненте указателе мыши |

2. Компонент TSavePictureDialog

Модуль – ExtDlgs

Описание:

Компонент TSavePictureDialog отображает модальное диалоговое окно для выбора имени и сохранения графических файл. Этот компонент похож на компонент TSaveDialog, за исключением того, что он включает в себя квадратное окно предварительного просмотра. Если выбранная картинка может отображаться с использованием компонента TPicture, то она отображается в области предварительного просмотра; поддерживаемые типы файлов включают в себя точечные рисунки Windows (.BMP), иконки (.ICO), Windows метафайлы (.WMF), и расширенный метафайл Windows (.EMF). Если выбранный файл не может быть отображен, в области предварительного просмотра появляестя надпись “(None)”.

Иерархия компонента:

TObject -> TPersistent -> TComponent -> TCommonDialog -> TOpenDialog -> TOpenPictureDialog -> TSavePictureDialog

Пример использования данного компонента в программе приведен на рисунке 17.

procedure TMainForm.mmSaveClick(Sender: TObject);

begin

if SavePictureDialog.Execute then Charter.SaveToFile(SavePictureDialog.FileName);

end;

Основные свойства, методы и события для этого компонента приведены в таблицах 8,9 и 10.

Таблица 8 – Основные свойства компонента ТGroupBox

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| property DefaultExt: String; | Устанавливает расширение файлов используемое по умолчанию |
| property FileName: TFileName; | Определяет имя последнего выбранного файла |
| property Filter: String; | Определяет маску файлов доступную в текущем диалоге |
| property Title: String; | Определяет заголовок текущего диалога |
| property InitialDir: String; | Определяет текущую директорию при открытии диалога |

Таблица 9 – Основные методы компонента ТGroupBox

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| function Execute: Boolean; override; | Открывает диалоговое окно |
| function GetStaticRect: TRect; virtual; | Определяет координаты зарезервированной области диалогового окна |

Таблица 10 – Основные события компонента ТGroupBox

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| property OnCanClose: TCloseQueryEvent; | Происходит когда пользователь пытается закрыть диалоговое окно |
| property OnFolderChange: TNotifyEvent; | Происходит при смене директории |
| property OnIncludeItem: TIncludeItemEvent; | Происходит перед тем, как выбранный файл добавляется в список |
| property OnTypeChange: TNotifyEvent; | Происходит когда типы файлов, отображаемых в диалоговом окне изменяются |

**4. ТЕСТОВЫЕ ВЫЗОВЫ ПРОГРАММЫ**

Ниже приведены примеры тестовых вызовов программы: рисунок 18 - пример отображения графика первого интеграла с переменным верхним пределом, рисунок 19 - примеры расчета первого и второго интеграла с заданными параметрами точности вычислений. На рисунке 20 представлен пример исключительной ситуации, вызванной неверным задание границ интегрирования. На рисунке 21 - пример вызова API-функции GetSystemInfo, отображающей информацию о текущей системе.

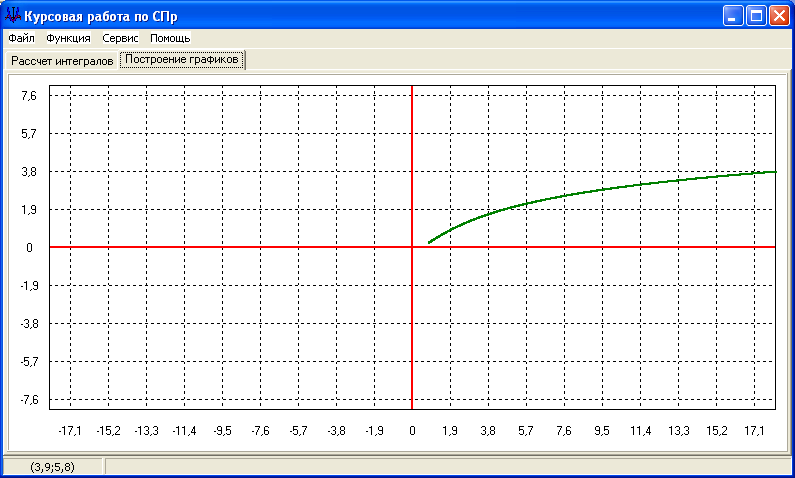


Рисунок 18 - Построение графика интеграла

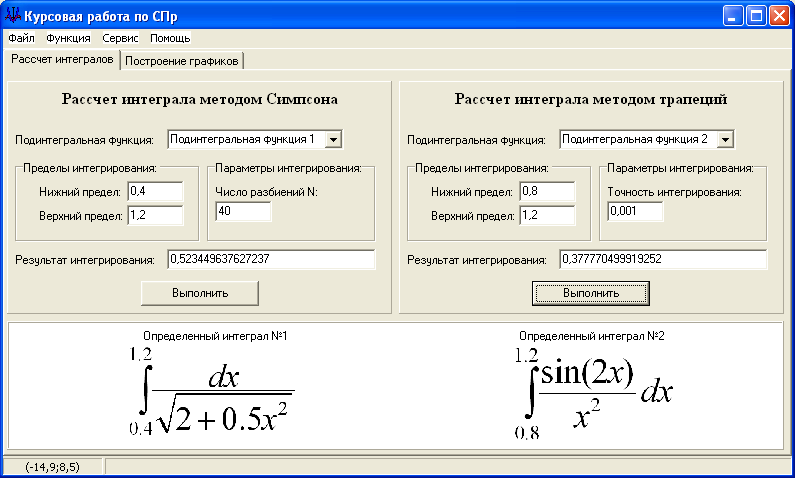


Рисунок 19 - Примеры расчета интегралов

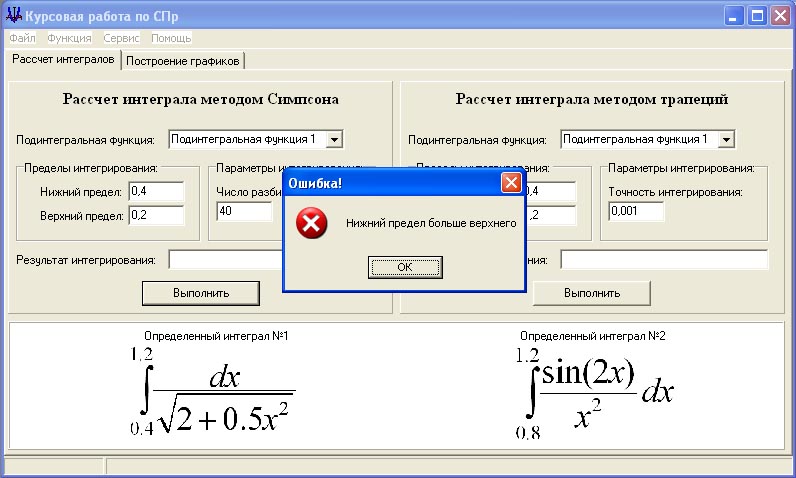


Рисунок 20 - Тестовый вызов исключительной ситуации

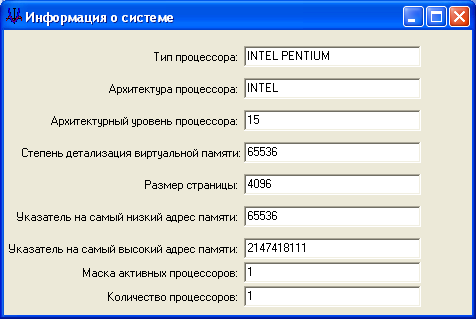


Рисунок 21 - Тестовый вызов API-функции GetSystemInfo

Для проверки правильности выполненных расчетов интегралов, представленных на рисунке 19, и отображения графика первого интеграла с переменным верхним пределом приведем пример вычисления интеграла и расчета графика выполненных в программе Mathcad Professional 2000, представленном на рисунке 22.

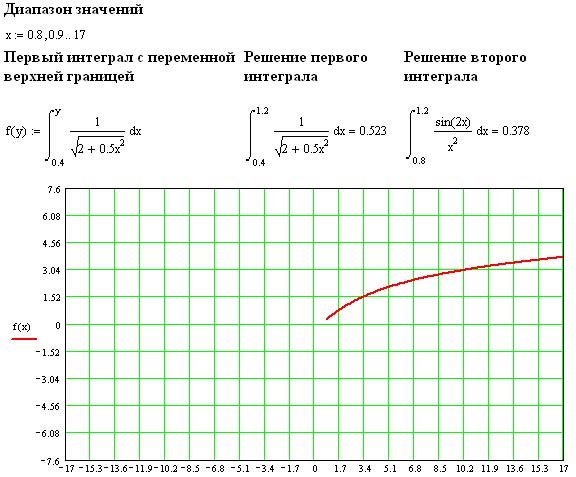


Рисунок 22 - Решение задачи в Mathcad

Как видно из рисунка, значения интегралов рассчитанных в разрабатываемой программе и в программе Mathcad Professional 2000 практически сходятся, что говорит правильности решения поставленной задачи.

**5. ОПИСАНИЕ РАЗРАБОТКИ СПРАВОЧНОЙ СИСТЕМЫ**

Для создания справочной системы я использовал программу DotCHM. Эта программа на лету создает справочные системы нескольких стандартов: 16-разрядная справка, 32-разрядная справка, chm-справка и справка для Windows CE. В DotCHM можно создавать полностью рабочий справочный файл, с поиском, обзором и содержанием, за считанные минуты. Все, что требуется это набрать текст справки.

Создав новый проект справочной системы, создаю стандартное содержание. Затем создаю список разбиений справки на пункты, заполняю эти пункты, добавляю нужные изображения, ссылки и т.д. Связываю все разделы с содержанием и компилирую.

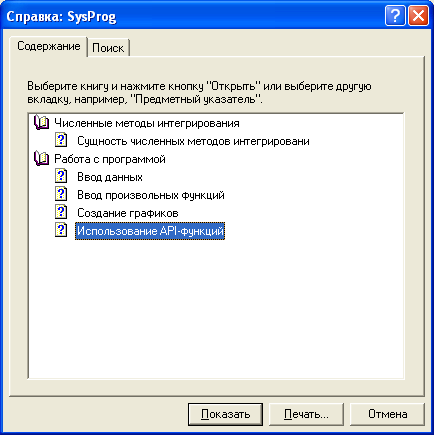


Рисунок 23 – Пример вызова справки

**ЗАКЛЮЧЕНИЕ**

Разработанная программа предоставляет возможности по вычислению интегралов с помощью численных методов с достаточно большой степенью точности, что подтверждается проведением аналогичных расчетов в профессиональной математической программе Mathcad Professional 2000. Кроме того, программа предоставляет возможность ввода произвольных функций, вычисления их интегралов и построения графиков функций. Данная программа может быть использована в учебных целях.

При разработке данной программы были приобретены навыки:

**-** разработки программ в среде программирования Delphi

**-** использования методики смешанного программирования

**-** использования динамической компоновки dll-библиотек

**-** программной реализации численных методов для вычисления определенных интегралов

**-** написания 16-ти разрядного Help-а под Windows

**ПЕРЕЧЕНЬ ССЫЛОК**

1. Турчак Л.И. Основы численных методов: Учеб. пособие. - М.: Наука, Гл. ред. физ. -мат. лит., 2002. - 320 с

2. Стив Т., Ксавье П. Delphi6 руководство разработчика.-М.: Вильямс, 2005г.- 1112с.

3. Дан Эпплман. Win32 API и Visual Basic для профессионалов.-М.: Питер, 2001г.- 1256с.

**ПРИЛОЖЕНИЕ А**

Листинг dll-библиотеки integrals.dll

library integrals;

uses

SysUtils,

Classes;

{$R \*.res}

type

IntegralFunction= function (x: real; var y: real): boolean;

function Simpson(MainFunction: IntegralFunction;a,b:real;n: integer; var Res: real): boolean;

var

s, h, x,y: real;

i: integer;

begin

Result:=true;

try

s:=0;

h:=(b-a)/n;

for i:=0 to n do

begin

x:=a+h\*i;

if (i=0) or (i=n) then if MainFunction(x,y) then s:=s+y else Result:=false else

if i mod 2 = 0 then if MainFunction(x,y) then s:=s+2\*y else Result:=false else

if MainFunction(x,y) then s:=s+4\*y else Result:=false;

end;

except

Simpson:=false;

end;

Res:=s\*h/3;

end;

function CountTrap(MainFunction: IntegralFunction; a,b, eps: real; var Res: real): boolean;

function PTrap(MainFunction: IntegralFunction; a,b:real; DivNo: Integer; var Res: real): boolean;

var

x1, x2,y1,y2,y3,dx,sum: real;

i: integer;

begin

PTrap:=true;

try

sum:=0;

dx:=(b-a)/DivNo;

for i:=1 to DivNo-1 do

begin

x1:=a+dx\*i;

if MainFunction(x1,y1) then sum:=sum+y1 else Raise EAbort.Create('');

end;

sum:=sum\*2;

if MainFunction(a,y1) and MainFunction(b,y2) then

sum:=sum+y1+y2 else Raise EAbort.Create('');

Res:=sum\*dx/2;

except

PTrap:=False;

end;

end;

{-----}

var

T: integer;

i1, i2: real;

begin

countTrap:=true;

try

if PTrap(MainFunction,a,b,2,i1) and PTrap(MainFunction,a,b,4,i2) then

begin

T:=4;

while abs(i1-i2)>eps do

begin

if PTrap(MainFunction,a,b,t,i1) and PTrap(MainFunction,a,b,t\*2,i2) then t:=t\*2

else EAbort.Create('');

T:=T\*2;

end;

end else EAbort.Create('');

Res:=(i1+i2)/2;

except

CountTrap:=false;

end;

end;

exports Simpson index 1;

exports CountTrap index 2;

begin

end.

**ПРИЛОЖЕНИЕ Б**

Листинг dll-библиотеки functions.dll

library functions;

uses

SysUtils,

Classes;

{$R \*.res}

function Integral1Function(x: real; var y: real):boolean;

begin

Result:=true;

try

y:=1/sqrt(2+0.5\*x\*x)

except

result:=false;

end;

end;

function Integral2Function(x: real; var y: real):boolean;

begin

Result:=true;

try

y:=sin(2\*x)/sqr(x)

except

result:=false;

end;

exports Integral1Function;

exports Integral2Function;

begin

end.

**ПРИЛОЖЕНИЕ В**

Листинг модуля chart.pas

unit Chart;

interface

uses Graphics, SysUtils, Classes;

const

BorderTop=10;

BorderBottom=40;

BorderLeft=40;

BorderRight=10;

type

GraphFunction = function(x: real; var y: real): boolean;

TChartFunction = class

private

MainFunction: GraphFunction;

FColor: TColor;

FShow: boolean;

public

constructor Create(f: GraphFunction);

published

property Color: TColor read FColor write FColor;

property Show: boolean read FShow write FShow;

end;

TChart = class

private

functions: TStringList;

ChartBitmap: TBitmap;

MainCanvas: TCanvas;

bgBrush: TBrush;

gridPen: TPen;

AxisPen: TPen;

Width: integer;

Height: integer;

FScale: real;

FStep: real;

FShowGrid: boolean;

xx0, yy0: integer;

nsteps: integer;

x0, y0: integer;

procedure DrawGrid;

procedure DrawFunction(funct: TChartFunction);

procedure DrawFunctions;

procedure PutPixel(x,y: integer; Color: TColor);

procedure OutTextCenter(s: string; x, y: integer);

public

constructor Create(Canvas: TCanvas);

procedure AddFunction(f: TChartFunction; Ident: String);

function DelFunction(s: string): boolean;

procedure ReDraw;

procedure ShiftHor(value: integer);

procedure ShiftVer(value: integer);

procedure Rescale(xvalue, yvalue: integer; Maximize: boolean);

procedure GetXY(x,y: integer; var x1,y1: real);

procedure SaveToFile(filename: string);

published

property ShowGrid: boolean read FShowGrid write FShowGrid;

end;

implementation

{ TChart }

constructor TChart.Create(Canvas: TCanvas);

begin

functions:=TStringList.Create;

MainCanvas:=Canvas;

ChartBitmap:=TBitmap.Create;

bgBrush:=TBrush.Create;

bgBrush.Color:=clWhite;

gridPen:=TPen.Create;

gridPen.Style:=psDot;

AxisPen:=TPen.Create;

AxisPen.Width:=2;

AxisPen.Style:=psInsideFrame;

AxisPen.Color:=clRed;

Width:=MainCanvas.ClipRect.Right-MainCanvas.ClipRect.Left;

Height:=MainCanvas.ClipRect.Bottom-MainCanvas.ClipRect.Top;

ChartBitmap.Width:=Width;

ChartBitmap.Height:=Height;

FShowGrid:=true;

xx0:=40+trunc((width-50)/2);

yy0:=10+trunc((height-50)/2);

x0:=xx0;

y0:=yy0;

NSteps:=20;

FScale:=20;

FStep:=(width/fscale)/nsteps;

ReDraw;

end;

procedure TChart.DrawFunction(funct: TChartFunction);

var

x, y: real;

x1,y1,x2,y2: integer;

a,b: boolean;

f: GraphFunction;

begin

x:=-(x0/FScale);

a:=false;

f:=funct.MainFunction;

while (x0+x\*FScale)<(Width-BorderRight) do

begin

if f(x,y) then

begin

x1:=round(x0+x\*FScale);

y1:=round(y0-y\*FScale);

if (x1>BorderLeft) and (x1<width-BorderRight) and (y1>BorderTop) and (y1<height-BorderBottom) then

begin

PutPixel(x1,y1, funct.Color);

end;

end;

x:=x+0.01;

end;

end;

procedure TChart.OutTextCenter(s: string; x, y: integer);

begin

x:=x-round(MainCanvas.TextWidth(s)/2);

y:=y-round(MainCanvas.TextHeight(s)/2);

MainCanvas.TextOut(x,y,s);

end;

procedure TChart.DrawGrid;

var

x, y: integer;

k: real;

s: string;

begin

With MainCanvas do begin

Brush.Assign(bgBrush);

FillRect(ClipRect);

begin

if FShowGrid then

Pen.Assign(GridPen) else Pen.Style:=psClear;;

x:=x0 mod trunc(FScale\*FStep);

while x<(width-BorderRight) do

end;

procedure TChart.ReDraw;

begin

DrawGrid;

DrawFunctions;

end;

procedure TChart.ShiftHor(value: integer);

begin

x0:=x0+value;

ReDraw;

end;

procedure TChart.ShiftVer(value: integer);

begin

y0:=y0+value;

ReDraw;

end;

procedure TChart.AddFunction(f: TChartFunction; Ident: String);

begin

functions.AddObject(Ident,f);

ReDraw;

end;

procedure TChart.DrawFunctions;

var

i: integer;

begin

for i:=0 to functions.Count-1 do

if TChartFunction(functions.Objects[i]).Show then

DrawFunction(TChartFunction(functions.Objects[i]));

end;

begin

if x>BorderLeft then

begin

MoveTo(x,BorderTop); LineTo(x, Height-BorderBottom);

k:=(x-x0)/FScale;

s:= FloatToStrF(k,ffGeneral,4,2);

OutTextCenter(s, x, height-round(BorderBottom/2));

end;

x:=x+trunc(FStep\*FScale);

end;

y:=y0 mod trunc(FScale\*FStep);

while y<(height-BorderBottom) do

begin

if y>BorderTop then

begin

MoveTo(BorderLeft,y); LineTo(Width-BorderRight, y);

k:=(y0-y)/FScale;

s:= FloatToStrF(k,ffGeneral,4,2);

OutTextCenter(s,round(BorderLeft/2), y);

end;

y:=y+trunc(FStep\*FScale);

end;

end;

// юЄЁшёютър ъююЁфшэрЄэ√ї юёхщ

Pen.Assign(AxisPen);

if (x0>BorderLeft) and (x0<(width-BorderRight)) then

begin

MoveTo(x0, BorderTop); lineto(x0, height-BorderBottom-1);

end;

if (y0>BorderTop) and (y0<(Height-BorderBottom)) then

begin

Moveto(BorderLeft,y0); lineTo(width-BorderRight-1, y0);

end;

Pen.Assign(gridPen);

Pen.Style:=psSolid;

Brush.Style:=bsClear;

Rectangle(40, 10, Width-10, height-40);

Brush.Style:=bsSolid;

end;

end;

procedure TChart.PutPixel(x, y: integer; Color: TColor);

var

i, j: integer;

begin

with MainCanvas do

for i:=x to x+1 do

for j:=y to y+1 do

MainCanvas.Pixels[i,j]:=Color;

function TChart.DelFunction(s: string): boolean;

label 1;

var

i: integer;

begin

result:=false;

for i:=0 to functions.Count-1 do

if functions.Strings[i]=s then

begin

result:=true;

functions.Delete(i);

goto 1;

end;

1:

end;

procedure TChart.Rescale(xvalue, yvalue: integer; Maximize: boolean);

var

mx, my: Real;

begin

if xvalue\*yvalue<>0 then begin

mx:=(width-borderleft-borderright)/xvalue;

my:=(height-bordertop-borderbottom)/yvalue;

if Maximize then

if mx<my then FScale:=FScale\*mx else FScale:=FScale\*my

else

if mx<my then FScale:=FScale/mx else FScale:=FScale/my;

FStep:=(width/FScale)/NSteps;

ReDraw; end;

end;

procedure TChart.GetXY(x, y: integer; var x1, y1: real);

begin

x1:=(x-x0)/FScale;

y1:=(y0-y)/FScale;

end;

procedure TChart.SaveToFile(filename: string);

var

temporary: TPicture;

begin

temporary:=TPicture.Create;

temporary.Bitmap.Width:=width;

temporary.Bitmap.Height:=height;

Temporary.Bitmap.Canvas.CopyRect( Temporary.Bitmap.Canvas.ClipRect, MainCanvas, MainCanvas.ClipRect);

Temporary.SaveToFile(filename);

temporary.Destroy;

end;

{ TChartFunction }

constructor TChartFunction.Create(f: GraphFunction);

begin

MainFunction:=f;

Show:=true;end;end.