**Библиотеки DLL, способы создания и использования**

DLL (Dynamic Link Library – динамическое подключение библиотеки) файл это специализированный файл того или иного языка программирования внутри которой хранятся различные функции и процедуры, предназначенные для выполнения необходимых операций, которые имеют определённую структуру и могут использоваться в нескольких программах. Проблема возникновения dll библиотек связано с проблемой многоразового использования написанного кода в основном в операционной системе Windows. DLL библиотеки дают возможность сохранять различные модули в отдельные файлы и необходимое время подключать их к основной программе, нежели стыковать каждый раз эти модули с программами, которые во первых занимают много места, во вторых снижают скорость т.е. быстродействие программы. Есть 3 недостатка:

1. Сначала DLL были процедурами, сейчас некоторые разработчики используют их ещё и для хранения объектов, которые являются неудобными
2. Коды из dll файла выполняются в том же участке памяти, где и основная программа, поэтому программа и библиотека используют один и тот же стэк данных, сто иногда накладывает свои ограничения
3. Отсутствие контроля версии

Способы создания Delphi: file -> new -> other -> dll -> ok;

Способы вызова DLL библиотек статистический и динамический.

Статистический в самой программе:

var

Form1: TForm1;

procedure QuickSort(var arr: iarray; left, right: Integer; ascending: Boolean);stdcall;

function IntegerNumber(arr : iARRAY; size : integer; mode : boolean = true):iarray; stdcall;

implementation

uses unit2;

procedure MergeSort; external 'Project1' name 'MergeSort';

function IntegerNumber; external 'Project1' name 'IntegerNumber';

Динамический способ вызова:

procedure TForm1.Button2Click(Sender: TObject);

var h : THandle;

func : procedure (var arr: iarray; left, right: Integer; ascending: Boolean);stdcall;

begin

h := LoadLibrary(pchar('Project1.dll'));\\ здесь имя dll файла

@sfun := GetProcAddress(h, pchar('StringSort'));

В самой библиотека нужно писать

function FloatNumber(arr : iARRAY1; size : integer; mode : boolean = true):iarray1; stdcall;

procedure Merge(var arr: sArray; left, middle, right: Integer; ascending: Boolean);

exports FloatNumber;

**Виды форм. Способы создания и использования**

Разновидности форм определяются свойством  FormStyle:

1. Стиль fsNormal определяет обычную форму, использующуюся для решения различных задач. Главная форма может иметь такой стиль.
2. Стиль fsMDIChild.
3. Cтиль fsMDIFormэ
4. Стиль fsStayOnTop - данный стиль преднзначен для окон, которые всегда должны располагаться над всеми другими окнами программ. Этот стиль используется в исключетельных случаях, когда окно содержит что-то, требующее повышенного внимания пользователя.

2,3 используются для создания MDI (Multi Document Interface)-приложений. Это стиль предполагает создание главного окна (его обычно называют рамочным), внутри которого по мере надобности появляются дочерние окна. Эти окна не могут выходить вне своего родителя - рамочного окна.  
Современные многооконные приложения стоятся в стиле SDI (Single Document Interface), который не  накладывает ограничений на положение и размеры вспомогательных форм, каждая из которых при необходимости может иметь свое главное меню.

File - New - Form  
По умолчанию главной формой является Form1. Можно указать любую форму, окно которой станет главным. Для этого выбираем команду Project - Options. Из списка MainForm выбираем главную.  
Каждое следующее окно будет появляться только после обращения к его методам Show() и ShowModal().  
Главное окно обязательно должно знать о существовании другого окна, что достигается ссылкой в предложении uses (Например, uses Unit2).  
Либо через главное меню: File - Use Unit.  
После этого подключения можно открыть окно второй формы, командой Form2.Show().  
Для переключения между формами используется команда ViewForm либо комбинация Shift + F12.  
При вызове метода Show второе окно появляется на экране и работает одновременно с первым, поэтому управление передается сразу оператору, стоящему за обращением к этому методу. Такие окна называются немодальными.   
ShowModal создает модальное окно, которое полностью берет на себя дальнейшее управление программой. Поэтому оператор за обращением к ShowModal вызывающей части программы получит управление только после закрытия модального окна. Модальные окна требуют от пользователя принятия какое-либо решения. В момент закрытия диалога модальное окно должно поместить число, соответствующее решению пользователя в свое свойство - ModalResult.   
Некоторые стандартные кнопки: Yes, No, Cancell автоматически выполняют эти действия (помещают нужное число в ModalResult и автоматически закрывают окно). Вызывающая программа получает значения ModalResult, как значение функции ShowModal, которое можно проанализировать. Модальные окна могут быть созданы с помощью функций ShowModal или InputBox.

Модальное окно и диалоговое окно - это два разных типа окон в пользовательском интерфейсе. Вот их основные отличия:

Модальное окно:

1. Блокирует доступ к другим окнам приложения, пока оно открыто. То есть, пользователь не сможет переключиться на другие окна или выполнять другие действия в приложении, пока модальное окно открыто.

2. Обычно используется для получения информации или ввода данных, которые требуют немедленного внимания пользователя.

3. Модальные окна могут быть созданы с помощью функций ShowModal или InputBox.

procedure TForm1.Button1Click(Sender: TObject);

var

InputString: string;

Begin InputString := InputBox('Ввод данных', 'Введите ваше имя:', ''); ShowMessage('Привет, ' + InputString + '!');end;

**Диалоговые окна способы использования**

Диалоговое окно - это специальное окно, которое появляется поверх основного интерфейса программы и запрашивает у пользователя определенную информацию или предлагает выбор определенных действий. Диалоговые окна могут быть использованы для ввода текста, выбора опций, отображения сообщений об ошибках или предупреждений, подтверждения действий и т. д. Они обычно блокируют взаимодействие с основным интерфейсом программы, пока не будут закрыты, чтобы пользователь мог сконцентрироваться на выполнении задачи, предложенной диалоговым окном.

Простое диалоговое окно с кнопкой

Синтаксис:

ShowMessage (const Msg: string);

Пример:

ShowMessage('123');

## Диалоговое окно с заголовком Acaption

**Синтаксис:**

InputBox (const Acaption, APromt, ADefault: string): string;

**Пример:**

InputBox ('Acaption','APromt','ADefault');

## Окно сообщения Msg типа DlgType

**Синтаксис:**

MessageDlg(const Msg:string; AType:TMsgType; AButtons: TmsgDlgButtons; HelpCtx:Longint): Word;

Окно сообщения Msg типа DlgType с набором кнопок Buttons, и системой справки HelpCtx.

**Пример:**

var

buttonSelected : Integer;

begin

buttonSelected := MessageDlg('Confirmation',mtError, mbOKCancel, 0);

if buttonSelected = mrOK then ShowMessage('OK');

if buttonSelected = mrCancel then ShowMessage('Cancel');

end.

## Простое окно сообщение с одной кнопкой

**Синтаксис:**

ShowMessageFmt(const Msg:string; Params: array of const);

Отображает простое окно сообщения, формируемого из параметров Params строкой форматирования Msg.

**Пример:**

ShowMessageFmt('%s', ['Hi']);

## Диалоговое окно сообщения Text с заголовком Caption

**Синтаксис:**

Aplication.MessageBox (Text, Caption:PChar; Flags:Longint): Integer;

Создает диалоговое окно сообщения Text с заголовком Caption. Параметр Flags определяет пиктограмму, набор кнопок и другие характеристики.

**Пример:**

var

ret: integer;

...

ret := Application.MessageBox(PChar('Вы хотите сохранить изменения?'),'Подтверждения',MB\_YESNOCANCEL + MB\_ICONQUESTION);

if ret = IDYES then ShowMessage('YES') //если да

else if ret = IDNO then close // если нет

else exit; // если отмена

На странице **Dialogs***Палитры компонентов* содержатся следующие компоненты, реализующие стандартные диалоговые окна:

* **openDialog** (выбор открываемого файла);
* **saveDialog** (выбор сохраняемого файла);
* **openPictureDialog** (выбор открываемого графического файла);
* **savePictureDialog** (выбор сохраняемого графического файла);
* **FontDialog** (настройка параметров шрифта);
* **ColorDialog** (выбор цвета);
* **PrintDialog** (вывод на принтер);
* **PrinterSetupDialog**(выбор принтера и настройка его параметров);
* **FindDialog** (ввод строки текста для поиска);
* **ReplaceDialog** (ввод строк текста для поиска и для замены);
* **PageSetupDialog** (установка параметров страницы).

**Использование возможностей реестра**

Это системная база данных в ней хранятся все необходимые значения и параметры самой операционной системы и всех программ используемых в ней. Чтобы открыть реестр необходимо нажать сочетание клавиш windows + R и ввести RegEdit. В появившемся окне окно разделено на 2 части в левой части показывает дерево или структура всех разделов ключей и подключей, а в правой части показывает список всех параметров и их значений для выбранных в левой части ключа или подключа. Сам реестр состоит из 5 разделов:

1.HKEY\_CLASSES\_ROOT-Он содержит информацию о зарегистрированных типах файлов, ассоциирующих программы, расширениях файлов, объектах OLE и других связанных с этими типами файлов данных.( Объект OLE (Object Linking and Embedding) - это технология, разработанная компанией Microsoft, предназначенная для обмена данными между приложениями. Она позволяет встраивать объекты из одного приложения в другое, а также создавать связи между объектами)

2.HKEY\_CURRENT\_USER – содержит индивидуальные установки для каждого пользователя, зарегистрированного в системе

3.HKEY\_LOCAL\_MACHINE – содержит аппаратные и программные установки необходимые для функционирования оборудования и программ. Данный раздел также хранит конфигурацию Windows

4.HKEY\_USES – содержит установки пользователей и соответствующие конфигурационные данные такие как : цвет окон, обои ,заливка и т.д.

5.HKEY\_USES\_CONFIG – содержит информацию о текущем аппаратном профиле, если пользователь не использует аппаратный профиль, то данный раздел содержит установки window по умолчанию

Для работы с реестром существует в основном 2 категории:

1.Функции из библиотеки API

2.Функции из библиотеки Registry разработанные для Delphi

API:

где key - раздел , subkey – подраздел

если возвращаемое значение ERROR SUCCES то всё успешно

1. RegCreateKey(key:Hkey; SubKey:Pchar; var:Result:Hkey):LongInt - для создания подраздела

2. RegOpenKey(key:Hkey; Subkey:pchar; var Result:Hkey):longint – для открытия раздела

3. RegCloseKey(key:Hkey):longint – для закрытия раздела

4. RegDeleteKey(key:Hkey; Subkey:pchar) – для удаления раздела

5. RegEnumKey(key:Hkey; index:longInt: buffer:pchar; cb:longInt):longint – для получения имён всех подразделов для заданного раздела, index-счётчик на количество вызовов функции, cb – размер буфера

6. RegQueryValue(key:Hkey; SubKey:Pchar; value:pchar; var cb:longint)longint – возвращает текстовый строки связанной с ключом и подключом

7. RegSetValue(key:Hkey; SubKey:Pchar; ValueType:longint; value:pchar; cb:longint):longint – для установки нового значения ключа

TRegistry: существует отдельный класс в виде библиотеки для работы с реестром в языке делфи. Все объекты, реализуемые для работы с реестром должны объявляться этим же типом. Чтобы использовать данный класс нужно в модулях подключить Registry

Var REG: TRegistry

REG:= TRegistry.Create

REG:= TRegistry.Destroy

Свойства:

1. RootKey : HKEY - это свойство возвращает значения типа HKEY допуская чтение и запись, кроме того определяет корневой ключ реестра.

Допустимыми значениями это свойства являются: HKEY\_CLASSES\_ROOT,

HKEY\_CURRENT\_USER , HKEY\_LOCAL\_MACHINE,

HKEY\_USERS , HKEY\_CURRENT\_CONFIG

2. CurrentPath : string - возвращает путь к текущему ключу

3. LazyWrite : Boolean - "ленивая запись", по умолчанию возвращает true и оно означает что на момент возврата из функции закрытие ключа не

гарантируется что изменения в нем записаны в реестр. Если false то на момент возврата из функции закрытия ключа гарантируется что все изменения уже записаны в реестр.

CurrentKey : HKEY - возвращает открытый в настоящее время ключ реестра

Методы: (func - функция, proc - процедура)

1. proc CloseKey - для закрытия ключа реестра

2. func CreateKey(key:string):boolean; - Для создания ключа

'\.Default\Software\MyProgram' - пример значения key (до MyProgram должен быть существующий путь иначе ошибка)

3. func DeleteKey(key:string):boolean; - удаляет ключ

4. func DeleteValue(name:string):boolean; - удаляет параметр с заданным именем текущего ключа. Для этого ключ заранее должен быть открытым.

5. func GetDataInfo(valuename:string, var value:TRegDataInfo):boolean; - (2-ой параметр возвращаемый) для получения информации о параметре заданной первым аргументом текущего ключа, информация типа размер… если удалось получить true в противном случае false

TRegDataType=(rdUnknown, rdString, rdExpandString, rdInteger, rdBinary)

TRegDataInfo=record RegData: TRegDataType; запись состоит из полей

DataSize: Integer;end;

GetDataSize, GetDataType

6. func GetDataSize(valuename:string):integer; - для получения размера параметра с заданным именем текущего ключа в байтах

7. func GetDataType(valeuname:string):TRegDataType; - для получения типа текущего ключа с заданным именем данная функция возвращает одно изэтих значений: rdUnknown,rdString,rdExpandString,rdInteger,rdBinary

8. func GetKeyInfo(var valuename:TRegKeyInfo):boolean; - для получения информации о ключе. True если получили информация о ключе, в противном случае false. Значением параметра функции является заполненная следующая структура

TRegKeyInfo=record

MaxSubKeyLen:integer; // кол-во подключей

NumValues:integer; // кол-во параметров

MaxValueLen:integer; // максимальная длина имени параметра

MaxDataLen:integer; // максимальная длина данных

FileTime:TFileTime; // время последней записи ключа

end;

9. proc GetKeyNames(str:TStrings); - заполняет указанный список str списком подключей текущего ключа

10. proc GetValueNames(str:TStrings); - заполняет указанный список str списком параметров текущего ключа

11. func HasSubKeys:boolean - возвращает true если текущий ключ имеет подключи и false в противном случае

12. func KeyExists(key:string):boolean; - возвращает true если ключ key существует и false в противном случае

13. fucn LoadKey(key,filename:string):boolean; - создает ключ key и загружает в него данные из файла с именем filename, true при успехе и тд

14. proc MoveKey(oldname,newname:string,delete:boolean); - предназначена для копирования или перименования ключа. Копирует все из oldname в newname и после копирования анализируеться delete. Если его значения true то ключ oldname уничтожаеться со всем содержимым,если false то ключ старые версии oldname копируются но не уничтожаются

15. func OpenKey(key:string,cancreate:boolean):boolean; - предназначена для открытия ключа, true если удалось открыть ключ иначе false, если cancreate = true при отсутствие ключа он его создаст, а при значение false нет

16. func OpenKeyReadOnly(key:string):boolean; - открывает ключ в режиме "только для чтения". true если удалось открыть в противном случае false

READ

17. func ReadBinaryData(name:string,var buffer,buffsize:integer):integer; - для чтения данных двочных параметров с заданным именем для текущего открытого ключа в buffer размера bufsize

18. func ReadBool(name:string):boolean; - для чтения значения параметра name типа boolean(логического типа).19. func ReadDate(name:string):TDate - для чтения значения параметра name типа дата

20. func ReadDateTime(name:string):TDateTime - для чтения значения параметра name типа дата/время

21. func ReadTime(name:string):TTime - для чтения значения параметра name типа время

22. func ReadFloat(name:string):Double - для чтения значения параметра name вещественного типа

23. func ReadInteger(name:string):integer - для чтения значения параметра name целого типа

24. func ReadString(name:string):string - для чтения значения параметра name строкового типа

25. func RegistryConnect(UNCname:string):boolean; - предназначена для подключения реестра сетового компютера

UNCname ---> [\\сет\_имя\_комп](file:///\\сет_имя_комп)

26. proc RenameValue(oldanme,newname:string); - для переименования параметра текущего ключа

27. func ReplaceKey(key,filename,backupfilename:string):boolean; - для замены места хранения ключа, в backupfilename будет копия ключа

28. func RestoreKey(key,filename:string):boolean; - для открытия указанного ключа и перезаписывания его данных и подключей даннымизагруженными из файла filename

29. func SaveKey(key,filename:string):boolean; - для сохранения всех параметров указанного ключа и всех его подключей в указанном файле filename

30. func UnLoadKey(key:string):boolean; - удаляет отдельные файлы ключа из реестра

31. func ValueExists(name:string):boolean; - предназначен для проверки существования параметра с заданным именем name в текущем ключе

32. proc WriteBinaryData(name:string,var buffer, bufsize:integer);

33. proc WriteDate(name:string,date:TDate);

34. proc WriteDateTime(name:string,date:TDateTime);

35. proc WriteTime(name:string,date:TTime);

36. proc ReadFloat(name:string,data:double);

37. proc WriteInteger(name:string,data:integer);

38. proc WriteBool(name:string,data:boolean);

39. proc WriteString(name:string,data:string);

40. proc WriteExpandString(name:stringm data:TStrings);

41. proc WriteCurrency(name:string,data:currency);

**Процедуры и функции для работы с канвой**

Является объектом, который используется в компонентах контейнера и при его помощи можно использовать классы для работы с графикой и создавать новые компоненты.

TFont – для настройки параметров шрифта

Name

Size

Color

Charset

Style

TPen – для настройки параметров карандаша

Color

Mode(для определения способа взаимодействия линии с фоном)

pmBlack

pmWhite

pmNop - цвет фона не меняется, линии невидны

pmNot-инверсия цвета фона(цвет будет манятся на противоположный)

pmCopy – цвет линии определяется цветом кисти

pmNotCopy – инверсия цвета кисти

pmMergePenNot – Комбинация цвета кисти и инверсного цвета фона

pmMaskPenNot – комбинация общих цветов для инверсных цветов кисти и фона

pmMergeNotPen – комбинация инверсных цвета кисти и фона

pmMaskNotPen – комбинация общих цветов для инверсных цветах кисти и фона

pmMerge – комбинация цветов кисти и фона

pmNotMerge – инверсия цветов кисти и фона

pmMask – общие цвета кисти и фона

pmNotMask – инверсия общих цветов кисти и фона

pmXor – объединение цветов кисти и фона операцией Xor(исключающее или )

pmNotXor – инверсия объединения цветов кисти и фона операцией Xor

**Style** – предназначен(intended) для определения стиля линии

psSolid –

psDash – пунктир

psDot – точка

psDashDot – тире точка

psDashDotDot – тире точка точка

psClear – невидимая линия

psInsideFrame - рисует внутри опр. области

Width – для установки толщины линии в пикселях

TBrush – для настройки параметров кисти

BitMap – предназначен для использования растровых изображений для заполнения закрашиваемой области

Color – для указания цвета кисти

Style – чтобы задавать стиль

* bsSolid – область полностью закрашивается
* bsClear – не заращивается
* bsHorizontal – область закрашивается горизонтальными линиями
* bsVertical – вертикальными
* bsFDiagonal – диагональными
* bsBDiagonal – побочными диагоналями
* bsCross – область заращивается сеткой
* bsDiagCross – область закрашивается сеткой из диагональных линий

TCanvas – обладает следующими свойствами

* Brush – закраска
* ClipRect – для определения текущих размеров области нуждающихся в отрисовке
* CopyMode – устанавливает способ взаимодействия растрового изображения с цветом фона
* cmBlackness
* cmDestInvert
* cmMergeCopy
* cmMergePaint
* cmNotSrcCopy
* cmNotSrcErase
* cmPatCopy
* cmPatInvert
* cmPatPaint
* cmSrcAnd
* cmSrcCopy
* cmSrcErase
* cmSrcInvert
* cmSrcPaint
* cmWhitness
* Font – доступ к фону
* Pen – доступ к карандашу
* PenPos – для опр. текущего положения карандаша в пикселях относительно левого верхнего угла Канвы
* Pixels[\_ , \_] – массив пикселов канвы

Методы Canvas {pr – процедура, fn - функция}

* pr Arc(x1,y1,x2,y2,x3,y3,x4,y4) – чертить дугу внутри прямоугольника с заданными координатами и начало и конец дуги будет находиться на пересечении эллипса и луча проведенной из его центра заданной точки
* pr Chord – чертит сегмент эллипса внутри прямоугольника с заданными координатами х1,у1,х2,у2 и начало и конец сегмента является пересечение эллипса и луча проходимые через центр эллипса
* Pr CopyRect (Dest:TRect, Canvas:TCanvas, Source:TRect) – копируют изображение Source канвы Canvas в участок Dest(destination) текущей канвы
* pr Draw(х,у,graph:TGraphic) – осуществляет отрисовку графического объекта заданный его третим параметром так чтобы левый верхний угол объекта расположился в точке (х,у)
* pr DrawFocusRect(rect:TRect) – отрисовывает прямоугольник с помощью операции Хor поэтому повторная отрисовка уничтожает ранее вычерченный прямоугольник
* pr Ellipse(x1,y1,x2,y2) – чертит эллипс в охватывающем прямоугольнике с координатами (х1,у1,х2,у2) заполняет внутреннее пространство текущей кистью
* pr FillRect – заполняет текущей кистью прямоугольный область Rect включая её левую и верхнюю границы, но исключая правую и нижнюю
* pr FloodFill(x,y,color,fillstyle) – заливка канвы. Заливка начинается с точки (х,у) и распространяется во все стороны от неё
* pr FrameRect(rect) – отчерчивает границы прямоугольника rect толщиной в один кисть без заполнения внутренней части
* pr LineTo(x,y) – чертит линию от текущего положения карандаша до точки (x,y)
* pr MoveTo(x,y) – перемещает карандаш в (х,у)
* pr Pie(x1,y1,x2,y2,x3,y3,x4,y4) – рисует сектор эллипса в охватывающем прямоугольника с данными началом и концом.
* pr Polygon(points:array of TPoint) – вычерчивает карандашом многоугольник по точкам заданным в виде массива
* pr Polyline – тоже самое что и Polygon но область не закращивается
* pr Rectangle(x1,y1,x2,y2) – вычерчивает и заполняет прямоугольник
* pr Refresh – устанавливает в канве параметры шрифта карандаша и кисти по умолчанию
* pr RoundRect – вычерчивает и заполняет прямоугольник с скруглёнными углами
* pr StretchDraw(rect, graphic) – вычерчивает и масштабирует графический объект так чтобы он полностью занял rect
* fn TextExtent(text: string):TSize – возвращает ширину и высоту прямоугольника охватывающего текстовую строку text
* fn TextHeight (text:string):Integer – высота прямоугольника охватывающая текстовую строку
* fn TextWidth(text:string):Integer – ширина прямоуг.
* pr TextOut(x,y,text) – верхний левый угол прямоугольника
* pr TextRect(rect, x,y,text) – выводит текст внутри прямоугольной области rect, так чтобы левый верхний угол прям. Охватывающее текст распологался в (х, у).

Некоторые функции библиотеки API для отрисовки компоненты.

При создании новых компонент с использованием функции API можно воспользоваться некоторыми функциями которые дают возможность получить доступ к конструкторам компонент или освободить их. Для графических компонентах эти операции выполняются через команду Canvas.Handle. Для оконных компонентов, где свойство Canvas не доступен можно использовать функцию GetDC(Handle), где

Var dc:HDC;

Dc = GetDC(Handle);

ReleasDC(Handle,dc) – удаляет.

Функции для отрисовки и закраски

* fn OffsetRect(rect,x,y):Boolean смещает прямоугольник rect на х пикселов в право и у пикселов вниз.
* fn InflateRect(rect,x,y) – увеличивает ширину и высоту прямоугольника rect соответсвенно х и у
* fn DrawFrameControl (dc,rect,utype,ustate) : boolean – отрисовывает рамку компоненты.
* DFC\_BUTTON – кнопка
* DFC\_CAPTION – кнопка в строке заголовка
* DFC\_MENU – меню
* DFC\_SCROLL – полоса прокрутки

DFC\_BUTTON

* DFCS\_BUTTON3STATE – трёхмерная кнопка
* DFCS\_BUTTONCHECK – флажок
* DFCS\_BUTTONPUSH – обычная кнопка
* DFCS\_BUTTONRADIO – переключатель
* DFCS\_BUTTONRADIOIMAGE – изображение переключателя
* DFCS\_BUTTONRADIOMASK – маска переключателя

DFC\_CAPTION

* DFC\_CAPTIONCLOSE – кнопка закрытия окна
* DFC\_CAPTIONHELP – кнопка вызова справки
* DFC\_CAPTIONMIN – кнопка свертывания
* DFC\_CAPTIONMAX - развертывания
* DFC\_CAPTIONRESTORE – восстановление размеров

DFC\_MENU

* DFC\_MENUARROW – указатель подменю
* DFC\_MENUBULLET – указатель зависимого меню
* DFC\_MENUCHECK – флажок меню

DFC\_SCROLL

* DFC\_SCROLLCOMBOBOX – полоса прокрутки в Combobox
* DFC\_SCROLLDOWN – кнопка прокрутки вниз
* DFC\_SCROLLUP – прокрутка вверх
* DFC\_SCROLLRIGHT – прокрутка в право
* DFC\_SCROLLLEFT – прокрутка в лево
* DFC\_SCROLLSIZEGRIP – кнопка изменения размеров в правом нижнем углу окна.

TRect

* Left
* Top
* Bottom
* Right
* Height
* Width

Значения могут быть дополнены и проверены следующими флажками:

DFCS\_ADJUSTRECT – корректирует границы

DFCS\_CHECKED – флажок установлен

DFCS\_FLAT – кнопка имеет плоские границы

DFCS\_INACTIVE – кнопка неактивна

DFCS\_MONO – кнопка имеет монохронные границы

DFCS\_PUSHED – кнопка нажата

fn DrawText(dc, text, count, var rect, uFormat):Integer – выводит заданное по count символов текста text в прямоугольнике rect.

uFormat принимает следующие значения:

* DT\_BOTTOM – прижатие текста в нижнюю часть
* DT\_CALCRECT – вычисляет минимально необходимый для вывода прямоугольник но не выводит текст
* DT\_CENTER – выводит текст по центру по горизонтали

DT\_EDITCONTROL – дублирует свойство многосторонности поля и не выводит частично видимую последнюю строку

* DT\_ENDELLIPSIS – заменяет многоточие начальные символы текста так чтобы он полностью уместился в прямоугольнике
* DT\_PATHELLIPSIS – заменяет символы в середине текста без изменения начальных и конечных символов так чтобы текст уместился в прямоугольнике
* DT\_EXPANDTABS – заменяет символы табуляции восьмикратной средней шириной символа (находит среднюю ширину среди всех символов увеличивает в 8 раз и заменяет символы табуляции на него)
* DT\_EXTERNALLEDING – учитывается межстрочное расстояние при вычислении высоты строки
* DT\_LEFT – прижимает строку влево
* DT\_MODIFYSTRING – модифицирует строку при использовании значений DT\_ENDELLIPSIS, DT\_PATHELLIPSIS
* DT\_NOCLIP – не отсекает текст границами прямоугольника
* DT\_NOPREFIX – отключает обработку управляющих символов
* DT\_RIGHT – прижимает текст в право
* DT\_RTLREADING – печатает текст в обратном порядке
* DT\_SINGLELINE – выводит текст в виде одной строки игнорируя символы конца строки
* DT\_TABSTOP – устанавливает ширину символа табуляции
* DT\_TOP – прижимает текст к верху прямоугольника
* DT\_VCENTRE – центрирует текст по вертикали
* DT\_WORDBREAK – разрешает разрыв строк на границах слов

fn SetBkMode(dc, BakMode):Integer – устанавливает способ заполнения фона при выводе текста

BakMode :

* Opaque – перед тем выводить текст фон текста будет закрашена выбранном цветом кисти + штриховка
* Transparent – отменяет заливку(можно использовать для прозрачности)

fn SetBkColor(dc, color):color - устанавливает новый цвет фона

если равно CLR\_INVALID то успешно, если нет то неуспешно.

**Способы динамического добавления объектов**

   Динамически создаваемые компоненты - это компоненты, место в памяти под которые выделяется по мере необходимости в процессе работы приложения. Этим они и отличаются от компонентов, которые помещаются на Форму при проектировании приложения. Возможность создавать компоненты динамически это очень большое удобство для программиста. Например, можно создавать в цикле сразу много однотипных компонентов, формируя из них массив, которым в дальнейшем очень просто управлять.  
  
   Все компоненты, как объекты, имеют множество свойств, определяющих их работу. При установке компонента на Форму из палитры большинство этих свойств определяются системой Delphi автоматически. При создании **динамического компонента** программист должен описать и настроить их вручную. Посмотрим, как это делается.  
  
   Прежде всего, для появления динамически создаваемого компонента нужно выделить под него место в памяти. Выделением места в памяти компьютера под любой компонент занимается конструктор типа объекта этого компонента - метод **Create**. Для этого сначала нужно описать переменную нужного типа, а затем для выделения памяти воспользоваться методом Create. Метод Create имеет параметр **Owner**, определяющий так называемого "владельца" для создаваемого компонента.  
  
   Хотя на самом деле владелец нужен не для создания, а для уничтожения компонента. То есть, при уничтожении компонента-владельца происходит автоматическое уничтожение всех компонентов, у которых он указан в качестве владельца.  
   При обычной установке компонента из палитры система делает владельцем этого компонента Форму. Проще всего поступать так же. Однако можно указать в качестве владельца сам этот компонент, воспользовавшись в качестве параметра ключевым словом **Self**.  
  
   Далее. Когда компонент создан, то есть место в памяти под него выделено, можно задавать значения параметрам этого объекта. Прежде всего, это ещё один компонент, так называемый "родитель". Компонент-родитель будет отвечать за отрисовку нашего динамически создаваемого компонента. Это значит, что новый компонент появится в границах компонента-родителя.  
  
   Если компонент-владелец имеет тип **TComponent**, то есть может быть любым компонентом, то компонент-родитель уже имеет тип **TWinControl**. То есть это должен быть "оконный" компонент, умеющий принимать и обрабатывать сообщения от системы Windows. Это необходимо, так как компонент должен находиться в некоторой иерархии компонентов, принимающих и передающих сообщения от системы Windows. Нашему динамическому компоненту сообщения будут передаваться через компонент-родитель.  
  
   А некоторые компоненты вообще не умеют принимать сообщения от системы, и в процессе работы в этом случае ими также будет управлять компонент-родитель, например, Форма или Панель, на которой они находятся.  
  
   Естественно, компонент не может быть родителем для самого себя. Имя компонента-родителя просто присваивается свойству **Parent** создаваемого динамически компонента.

unit Unit1;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs,Vcl.StdCtrls;// обязательно нужно добавить модуль

type

TForm1 = class(TForm)

procedure FormCreate(Sender: TObject);

procedure FormClick(Sender: TObject);

private

procedure BTNClick(Sender: TObject);//добавляем событие

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

BTN : TButton;// объявление переменной и тип к которому она относится

BTN\_NAME\_INDEX, btn\_i, btn\_j, rasstoyanie :integer;// для добавления компанент в цикле

implementation {

procedure TForm1.BTNClick(Sender: TObject); // добавление события

begin

if (Sender is TButton) then // Если объект кнопка то

begin

ShowMessage('Кнопка нажата'); // Выводим сообщение на экран

end;

end;

}

{ Получение значений свойств динамического компонента.

procedure TForm1.BTNClick(Sender: TObject);

begin

if (Sender is TButton) then // Если объект кнопка то

begin

ShowMessage('Кнопка '+ (Sender as TComponent).Name+' нажата'); // Выводим сообщение на экран

end;

end;

}

procedure TForm1.BTNClick(Sender: TObject);

begin

if (Sender is TButton) then // Если объект кнопка то

begin

//ShowMessage('Кнопка '+ (Sender as TComponent).Name+' нажата'); // Выводим сообщение на экран

if FindComponent('BTN11')<>nil then // если компонент BTN11 найден то

ShowMessage('Кнопка BTN11 найдена');

(FindComponent('BTN11') as TButton).Caption:='Кнопка 11';// можно использовать если хотим поменять caption

end;

end;

{$R \*.dfm}

procedure TForm1.FormClick(Sender: TObject);

begin

BTN.Free; // для уничтожения компанент

freeAndnil(BTN);// 2 способ уничтожения

end;

procedure TForm1.Formcreate(Sender: TObject);

begin

// простое динамическое создание компаненты

btn := Tbutton.Create(Form1);

btn.Parent := Form1;// устанавливаем родителя, при удалениии родителя удаляется и компанента

BTN.Top := 120;

BTN.Left := 120;

btn.Width := 120;

btn.Height := 120;

BTN.OnClick := BTNClick; // подключаем обработчик

btn.Name := 'Fotima';// имя должно быть индивидуальным

{

Динамическое объявление компанент в цикле

rasstoyanie:=5; // Устанавливаем расстояние между кнопками

BTN\_NAME\_INDEX:=1; // Устанавливаем значение для первой кнопки

for btn\_i := 1 to 8 do // повторяем цикл 8 раз

begin

for btn\_j:=1 to 3 do

begin

BTN := TButton.Create(Form1); // Создаём компонент

BTN.Parent := Form1; // Указываем родителя (в данном случае эту форму)

// Задаём расположение кнопки

BTN.Top := 30\*btn\_i+rasstoyanie-30; // с верху 30

BTN.Left := 100\*btn\_j+rasstoyanie-100;; // слева 30

BTN.Caption := 'Кнопка'; // Устанавливаем надпись на кнопку

BTN.Name:='BTN'+IntToStr(BTN\_NAME\_INDEX); // Даем имя кнопке

BTN.Width:=100; // ширина 100

BTN.Height:=30; // высота 30

inc(BTN\_NAME\_INDEX); // прибавляем к переменной BTN\_NAME\_INDEX единицу

end;

end; }

end;

end.

**Способы рисования на канве**

Методы Canvas {pr – процедура, fn - функция}

pr Arc(x1,y1,x2,y2,x3,y3,x4,y4) – чертить дугу внутри прямоугольника с заданными координатами и начало и конец дуги будет находиться на пересечении эллипса и луча проведенной из его центра заданной точки

pr Chord – чертит сегмент эллипса внутри прямоугольника с заданными координатами х1,у1,х2,у2 и начало и конец сегмента является пересечение эллипса и луча проходимые через центр эллипса

Pr CopyRect (Dest:TRect, Canvas:TCanvas, Source:TRect) – копируют изображение Source канвы Canvas в участок Dest(destination) текущей канвы

pr Draw(х,у,graph:TGraphic) – осуществляет отрисовку графического объекта заданный его третим параметром так чтобы левый верхний угол объекта расположился в точке (х,у)

pr DrawFocusRect(rect:TRect) – отрисовывает прямоугольник с помощью операции Хor поэтому повторная отрисовка уничтожает ранее вычерченный прямоугольник

pr Ellipse(x1,y1,x2,y2) – чертит эллипс в охватывающем прямоугольнике с координатами (х1,у1,х2,у2) заполняет внутреннее пространство текущей кистью

pr FillRect – заполняет текущей кистью прямоугольный область Rect включая её левую и верхнюю границы, но исключая правую и нижнюю

pr FloodFill(x,y,color,fillstyle) – заливка канвы. Заливка начинается с точки (х,у) и распространяется во все стороны от неё

pr FrameRect(rect) – отчерчивает границы прямоугольника rect толщиной в один кисть без заполнения внутренней части

pr LineTo(x,y) – чертит линию от текущего положения карандаша до точки (x,y)

pr MoveTo(x,y) – перемещает карандаш в (х,у)

pr Pie(x1,y1,x2,y2,x3,y3,x4,y4) – рисует сектор эллипса в охватывающем прямоугольника с данными началом и концом.

pr Polygon(points:array of TPoint) – вычерчивает карандашом многоугольник по точкам заданным в виде массива

pr Polyline – тоже самое что и Polygon но область не закращивается

pr Rectangle(x1,y1,x2,y2) – вычерчивает и заполняет прямоугольник

pr Refresh – устанавливает в канве параметры шрифта карандаша и кисти по умолчанию

pr RoundRect – вычерчивает и заполняет прямоугольник с скруглёнными углами

pr StretchDraw(rect, graphic) – вычерчивает и масштабирует графический объект так чтобы он полностью занял rect

fn TextExtent(text: string):TSize – возвращает ширину и высоту прямоугольника охватывающего текстовую строку text

fn TextHeight (text:string):Integer – высота прямоугольника охватывающая текстовую строку

fn TextWidth(text:string):Integer – ширина прямоуг.

pr TextOut(x,y,text) – верхний левый угол прямоугольника

pr TextRect(rect, x,y,text) – выводит текст внутри прямоугольной области rect, так чтобы левый верхний угол прям. Охватывающее текст распологался в (х, у).

**Способы закраски областей на канве**

1. pr FillRect – заполняет текущей кистью прямоугольный область Rect включая её левую и верхнюю границы, но исключая правую и нижнюю

2. pr FloodFill(x,y,color,fillstyle) – заливка канвы. Заливка начинается с точки (х,у) и распространяется во все стороны от неё

3. pr Rectangle(x1,y1,x2,y2) – вычерчивает и заполняет прямоугольник

4. pr RoundRect – вычерчивает и заполняет прямоугольник с скруглёнными углами

1. Использование метода fillRect() или fill() для закраски прямоугольной области. Этот метод принимает координаты верхнего левого угла прямоугольника и его ширину и высоту.

2. Использование метода beginPath(), moveTo(), lineTo() и fill() для рисования произвольной фигуры и её закраски. Этот метод позволяет создавать любые формы, а затем закрашивать их.

3. Использование метода createLinearGradient() или createRadialGradient() для создания градиента, который можно использовать для закраски областей.

4. Использование метода createPattern() для создания шаблона, который можно использовать для заполнения област

**Способы вывода текста на канвеей**.

fn TextExtent(text: string):TSize – возвращает ширину и высоту прямоугольника охватывающего текстовую строку text

fn TextHeight (text:string):Integer – высота прямоугольника охватывающая текстовую строку

fn TextWidth(text:string):Integer – ширина прямоуг.

pr TextOut(x,y,text) – верхний левый угол прямоугольника

pr TextRect(rect, x,y,text) – выводит текст внутри прямоугольной области rect, так чтобы левый верхний угол прям. Охватывающее текст распологался в (х, у).

Из библиотеки API:

fn DrawText(dc, text, count, var rect, uFormat):Integer – выводит заданное по count символов текста text в прямоугольнике rect.

uFormat принимает следующие значения:

* DT\_BOTTOM – прижатие текста в нижнюю часть
* DT\_CALCRECT – вычисляет минимально необходимый для вывода прямоугольник но не выводит текст
* DT\_CENTER – выводит текст по центру по горизонтали

DT\_EDITCONTROL – дублирует свойство многосторонности поля и не выводит частично видимую последнюю строку

* DT\_ENDELLIPSIS – заменяет многоточие начальные символы текста так чтобы он полностью уместился в прямоугольнике
* DT\_PATHELLIPSIS – заменяет символы в середине текста без изменения начальных и конечных символов так чтобы текст уместился в прямоугольнике
* DT\_EXPANDTABS – заменяет символы табуляции восьмикратной средней шириной символа (находит среднюю ширину среди всех символов увеличивает в 8 раз и заменяет символы табуляции на него)
* DT\_EXTERNALLEDING – учитывается межстрочное расстояние при вычислении высоты строки
* DT\_LEFT – прижимает строку влево
* DT\_MODIFYSTRING – модифицирует строку при использовании значений DT\_ENDELLIPSIS, DT\_PATHELLIPSIS
* DT\_NOCLIP – не отсекает текст границами прямоугольника
* DT\_NOPREFIX – отключает обработку управляющих символов
* DT\_RIGHT – прижимает текст в право
* DT\_RTLREADING – печатает текст в обратном порядке
* DT\_SINGLELINE – выводит текст в виде одной строки игнорируя символы конца строки
* DT\_TABSTOP – устанавливает ширину символа табуляции
* DT\_TOP – прижимает текст к верху прямоугольника
* DT\_VCENTRE – центрирует текст по вертикали
* DT\_WORDBREAK – разрешает разрыв строк на границах слов

fn SetBkMode(dc, BakMode):Integer – устанавливает способ заполнения фона при выводе текста

BakMode :

* Opaque – перед тем выводить текст фон текста будет закрашена выбранном цветом кисти + штриховка
* Transparent – отменяет заливку(можно использовать для прозрачности)

fn SetBkColor(dc, color):color - устанавливает новый цвет фона

если равно CLR\_INVALID то успешно, если нет то неуспешно.

**Способы создания и использования INI-файлов**

Иногда при реализации программы приходиться использовать некоторые специальные возможности, которые обеспечивают сохранение каких-либо

определенных параметров программы, которые были настроены до его нового запуска. К таким возможностям относиться ini файлы. В некоторых моментах при разработке программ лучше использовать ini файл чем реестр ОС.

Данный способ дает возможность:

1. Редактировать или смотреть ini файлы можно в обычном блокноте (не нужны специализированные программы).

2. Если ini файл храниться в одной папке вместе с программой, то при переносе папки на другой компьютер настройки сохраняться.

3. Новичку в реестре можно запутаться или если он изменит что-нибудь другое.

INI файлы в основном имеют следующую структуру:

[секции]

параметры секции со значениями

Для работы с ini файлами в языке Delphi существует отдельный класс, реализованный в виде модуля с названием IniFiles а объект класса TIniFiles

Для работы с объектами класса TIniFiles существуют следующие функции:

Constructor Create('D:\MyProgramm\settings.ini'); //Создание экземпляра

WriteBool(Sec,Par,Val) //Для записи логических значений (название секций, название параметра, зачение)

WriteInteger(Sec,Par,Val); //Для записи целого числа (название секций, название параметра, зачение)

WriteString(Sec,Par,Val); //Для записи строкового значения (название раздела, название параметра, значение)

ReadSection(Sec:string, Values:TStrings); //Для прочтения параметров секции (название секций, значение параметра секции)

ReadSectionValues(Sec:string, Values:TStrings); //Для прочтения значений всех параметров секции

EraseSection(Sec:String); //Для удаление секции со всем содержимым (название секции)

ReadBool(Sec,Par:string, Default:boolean) :Boolean; //Для прочтения логического значения(название секции, параметр, значение)

ReadInteger(Sec,Par:string, Default:longint) :longint; //Для прочтения значения типа Longint из параметра заданного раздела

ReadString(Sec,Par:string, Default:string) :string; //Для прочтения значений строкового типа

Free //Для закрытия и освобождения памяти от экземпляра ini файла

Property Values[name:string]:string //Для доступа к существующему параметру с заданным именем

Для проекта необходимо создать файл с расширением ini

[Form]

left =100

top =100

width =1106

height =312

color =$000080FF

Внутри проекта

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, IniFiles, StdCtrls;

type

TForm1 = class(TForm)

Button1: TButton;

ColorDialog1: TColorDialog;

procedure FormCreate(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure Button1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

procedure TForm1.FormCreate(Sender: TObject);

var IFile : TIniFile;

begin

IFile:=TIniFile.Create(ExtractFilePath(Application.ExeName)+'fotima.ini');

Form1.left:=IFile.readInteger('Form','left',50);

Form1.Top:=IFile.readInteger('Form','Top',50);

Form1.Width:=IFile.readInteger('Form','Width',800);

Form1.Height:=IFile.readInteger('Form','Height',600);

Form1.Color:=StringTocolor(Ifile.readString('Form','color','clbtnFace'));

IFile.free;

end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);

var IFile : TIniFile;

begin

IFile:=TIniFile.Create(ExtractFilePath(Application.ExeName)+'fotima.ini');

IFile.WriteInteger('Form','left',Form1.left);

IFile.WriteInteger('Form','top',Form1.top);

IFile.WriteInteger('Form','width',Form1.width);

IFile.WriteInteger('Form','height',Form1.height);

IFile.writeString('form','color',colortostring(form1.Color));

IFile.free;

end;

procedure TForm1.Button1Click(Sender: TObject);

begin

if(colorDialog1.Execute)then

form1.color:=colorDialog1.color;

end;

end.

**Способы создания пользовательских компонент**

При создании компонент необходимо выполнить следующие основные этапы:

1. Выбор родительского класса – в этом этапе можно создать новую компоненту несколькими способами:

А. Изменение существующих компонент(TButton,TLabel,TEdit…..)

Б. Создание новых оконных компонент, то есть компонента будет видимой (TWinControl,TCustomControl)

В. Создание новых графических (не оконных) компонентов (TGraphicControl, TGraphic)

Г. Создание подклассов для элементов управления windows (компонент из API)

Д. Создание не визуальных компонент(TComponent)

2. Создание модуля для компонент

3. Создание свойств методов и событий

4. Тестирование и откладка компонент

5. Оформление компонент

Существует 2 способа для создания новых компонент:

1.File -> New -> Other -> Component

2.Component -> new Component

После выполнения одной из перечисленных команд появляется окно создания новых компонент. В нём необходимо выполнить следующие действия

Ancestor: выбрать родительский класс

Class Name: добавить имя класса новой компоненты при написании имени класса рекомендуется соблюдать стандарт классов для компонент (TBtnEllipse)

Palette Page: выбрать название группы на котором появляется создаваема компонента

Unit file name: выбрать file источник компонент

Search Path: если необходимо в поле search path добавить каталог

**Способы создания программ с использованием VCL**

VCL (Visual Component Library) - это набор компонентов, который предоставляет средства для создания графического интерфейса в Delphi. С помощью VCL вы можете создавать различные типы приложений, включая оконные приложения, консольные приложения, службы Windows и другие.

Вот некоторые способы создания программ с использованием VCL в Delphi:

1. Создание нового проекта: Вы можете создать новый проект в Delphi и выбрать тип приложения, который вы хотите создать (например, форма VCL или консольное приложение). Затем вы можете добавлять компоненты VCL на форму и программировать их поведение с помощью языка программирования Object Pascal.

2. Программирование событий: Вы можете привязывать обработчики событий к различным компонентам VCL, чтобы реагировать на действия пользователя или изменения в приложении. Например, вы можете написать код для обработки щелчка мыши на кнопке или изменения значения в поле ввода.

3. Создание пользовательского интерфейса: С помощью компонентов VCL вы можете создавать пользовательский интерфейс, включая формы, кнопки, поля ввода, таблицы, списки и другие элементы. Вы можете настраивать их внешний вид, поведение и взаимодействие с пользователем.

4. Работа с данными: VCL предоставляет компоненты для работы с различными типами данных, такими как базы данных, файлы, сетевые соединения и другие. Вы можете использовать компоненты для извлечения, сохранения и обработки данных в вашем приложении.

5. Создание пользовательских компонентов: Вы также можете создавать собственные пользовательские компоненты VCL, расширяя функциональность базовых компонентов или создавая собственные компоненты с нуля. Это позволяет вам создавать переиспользуемый код и упрощать разработку приложений.

В целом, Delphi с использованием VCL предоставляет мощные средства для создания разнообразных пр

VCL (Visual Component Library) - это библиотека компонентов пользовательского интерфейса для Delphi. Она содержит широкий набор компонентов, которые можно использовать для создания приложений с графическим интерфейсом пользователя (GUI). VCL была разработана для упрощения создания приложений в Delphi, позволяя программистам использовать предопределенные компоненты, такие как кнопки, текстовые поля, таблицы, списки и многие другие элементы пользовательского интерфейса. Кроме того, VCL предоставляет возможность создания собственных компонентов, что позволяет улучшить переиспользуемость кода и ускорить процесс разработки. VCL является одной из главных особенностей Delphi и обеспечивает его быстрое и эффективное создание приложений с графическим интерфейсом пользователя. Как использовать VCL в Delphi? Чтобы использовать VCL в Delphi, вам необходимо выполнить следующие шаги:

Создайте новый проект в Delphi, выбрав тип приложения, который соответствует вашим требованиям.

Откройте окно инструментов и выберите вкладку "Components" (компоненты). Выберите нужные компоненты из списка, перетащив их на форму вашего приложения.

Настройте свойства компонентов, чтобы они соответствовали вашим потребностям.

Добавьте обработчики событий для компонентов, если это необходимо.

Напишите код для обработки событий, который определит, что должно происходить при выполнении действия пользователем.

Пример использования VCL в Delphi

Создайте новый проект типа "VCL Forms Application".

На форму добавьте компоненты TLabel и TButton, перетащив их из панели инструментов на форму.

Настройте свойства компонентов, чтобы изменить текст на кнопке и метке.

Добавьте обработчик события OnClick для кнопки, дважды щелкнув на ней. Внутри обработчика напишите следующий код:

procedure TForm1.Button1Click(Sender: TObject);

begin

Label1.Caption := 'Hello, World!';

end;

Запустите приложение и нажмите на кнопку, чтобы изменить текст на метке.  
ограммных приложений с графическим интерфейсом. Сочетание визуального

TObject

|

-------------------------------------------------------------------------

| | | | |

Exception TInterfacePbject TStream TPersistant TComObject

|

-------------------------------------------------------------------------------------------------

| | | | |

TGraphicObject TGraphic TComponents TCollections TStrings

|

------------------------------------------------------------------------------------------

| | | | | |

TApplication TDataSet TMenu TControl TCommOnDialog TTimer

|

----------------------------

| |

TGreaphicControl TWinControl

|

------------------------------

| |

TScrollingWinControl TCustomControl

**Статистическое и динамическое связывание Dll библиотек**

Способы создания Delphi: file -> new -> other -> dll -> ok;

Способы вызова DLL библиотек статистический и динамический.

Статистический в самой программе:

var

Form1: TForm1;

procedure QuickSort(var arr: iarray; left, right: Integer; ascending: Boolean);stdcall;

function IntegerNumber(arr : iARRAY; size : integer; mode : boolean = true):iarray; stdcall;

implementation

uses unit2;

procedure MergeSort; external 'Project1' name 'MergeSort';

function IntegerNumber; external 'Project1' name 'IntegerNumber';

Динамический способ вызова:

procedure TForm1.Button2Click(Sender: TObject);

var h : THandle;

func : procedure (var arr: iarray; left, right: Integer; ascending: Boolean);stdcall;

begin

h := LoadLibrary(pchar('Project1.dll'));\\ здесь имя dll файла

@sfun := GetProcAddress(h, pchar('StringSort'));

В самой библиотека нужно писать

function FloatNumber(arr : iARRAY1; size : integer; mode : boolean = true):iarray1; stdcall;

procedure Merge(var arr: sArray; left, middle, right: Integer; ascending: Boolean);

exports FloatNumber;

**Структура пользовательских компонент**

1. \*\*Класс компонента (Component Class)\*\*: Каждый пользовательский компонент является классом, который наследуется от существующего класса компонента в VCL, такого как TComponent, TControl, TCustomControl и т. д. Этот класс определяет основное поведение и свойства компонента.

2. \*\*Свойства (Properties)\*\*: Компонент может иметь различные свойства, которые определяют его внешний вид, поведение и другие характеристики. Например, свойства могут включать цвет, размер, текст и другие параметры, которые можно настроить через дизайнер форм в среде разработки Delphi.

3. \*\*События (Events)\*\*: Компоненты могут генерировать события, на которые можно реагировать в коде приложения. Примерами событий могут быть клик мыши, изменение значения и другие действия, которые могут происходить во время работы приложения.

4. \*\*Методы (Methods)\*\*: Методы представляют собой функции или процедуры, которые определяют поведение компонента. Они могут использоваться для выполнения определенных действий, обработки данных или управления внутренним состоянием компонента.

5. \*\*Индексаторы (Indexers)\*\*: Индексаторы позволяют обращаться к элементам компонента по индексу, что особенно полезно для коллекций или контейнеров.

6. \*\*Публичные, защищенные и приватные члены\*\*: Класс компонента может содержать публичные, защищенные и приватные члены, определяющие доступность и об

unit MyCustomComponent;

interface

uses

Classes, Controls;

type

TMyCustomComponent = class(TCustomControl)

private

FCustomProperty: Integer;

procedure DoSomething;

protected

procedure Paint; override;

public

constructor Create(AOwner: TComponent); override;

property CustomProperty: Integer read FCustomProperty write FCustomProperty;

procedure PublicMethod;

event OnCustomEvent: TNotifyEvent;

end;

implementation

constructor TMyCustomComponent.Create(AOwner: TComponent);

begin

inherited Create(AOwner);

// Инициализация компонента

end;

procedure TMyCustomComponent.DoSomething;

begin

// Реализация метода

end;

procedure TMyCustomComponent.Paint;

begin

// Реализация отрисовки компонента

end;

procedure TMyCustomComponent.PublicMethod;

begin

// Реализация публичного метода

end;

end.

асть видимости свойств, методов и событий.

**Установка собственных компонент**

Установка собственных компонентов в Delphi включает несколько шагов:

1. \*\*Создание компонента\*\*: Создайте новый проект компонента в среде разработки Delphi. Для этого выберите "File" -> "New" -> "Other" -> "Delphi Projects" -> "VCL" -> "Component". Затем выберите тип компонента, который вы хотите создать (например, "VCL Component" или "Custom Control"), и следуйте инструкциям мастера создания компонента.

2. \*\*Разработка компонента\*\*: Разработайте свой компонент, определив его свойства, методы, события и другие характеристики в соответствии с требованиями вашего приложения.

3. \*\*Сохранение компонента\*\*: После завершения разработки сохраните свой компонент в отдельном файле с расширением .pas. Например, MyCustomComponent.pas.

4. \*\*Установка компонента\*\*: Чтобы установить свой компонент в Delphi, откройте меню "Component" -> "Install Component". В появившемся диалоговом окне нажмите кнопку "Browse", выберите файл вашего компонента (MyCustomComponent.pas) и нажмите "Open". Затем нажмите "OK" для установки компонента.

5. \*\*Использование компонента\*\*: После успешной установки ваш компонент будет доступен для использования в других проектах. Вы можете найти его во вкладке "Tool Palette" в разделе "Installed" или "Samples".

6. \*\*Тестирование компонента\*\*: После установки компонента рекомендуется протестировать его, создав новый проект и добавив ваш компонент на форму. Затем протестируйте его поведение, свойства и события.

**Настройка пользовательских компонент**

Любой вновь создаваемый класс может содержать секции (разделы), определяемые зарезервированными словами **published**(опубликованные), **private** (закрытые),**protected**(защищенные), **public**(доступные) и **automated** (автоматизированные). Внутри каждой секции вначале определяются поля, а затем - методы и свойства.  
  
Секции определяют области видимости элементов описания класса.  
Секция **public** не накладывает ограничений на область видимости перечисляемых в ней полей, методов и свойств - их можно вызывать в любом другом модуле программы.  
  
Секция published также не ограничивает область видимости, однако в ней перечисляются свойства, которые должны быть доступны не только на этапе исполнения, но и на этапе конструирования программы (т. е. в окне Инспектора объектов). Секция **published** используется только при разработке нестандартных компонентов. Замечу, что среда Delphi помещает описания компонентов, вставленных в форму, в специальную секцию без названия, которая располагается сразу за заголовком класса и продолжается до первой объявленной секции. Эта секция - published. Программисту не следует помещать в нее собственные элементы описания класса или удалять из нее элементы, вставленные средой.  
  
**published**используется для написания компент и интеграции их в IDE Delphi, если мы пишем не компонент, то нам хватит и public.  
**private и protected** используются для сокрытия внутренностей реализации класса (один из принципов ООП. класс - черный ящик).  
  
Секция **private** сужает область видимости до минимума: закрытые элементы описания доступны только внутри методов данного класса и подпрограммах, находящихся в том же модуле, где описан класс. Элемент, объявленный в секции private, становится недоступным даже ближайшим потомкам класса, если они размещаются в других модулях.  
  
Секция **protected** доступна только методам самого класса, а также любым его потомкам, независимо от того, находятся ли они в том же модуле или нет.  
**protected** позволяет наследникам класса всетаки переопределить поведение класса.  
Наконец, секция **automated** используется только для объявления свойств и методов, которые будут добавлены к так называемому интерфейсу OLE-объектов Автоматизации; область видимости членов этой секции не ограничена.

Настройка пользовательских компонентов означает определение и настройку свойств, событий и методов пользовательских компонентов для дальнейшего использования в приложениях.

Когда вы создаете пользовательский компонент, вы можете определить свойства, которые позволяют настраивать его внешний вид, поведение и другие характеристики. Например, вы можете добавить свойство `Color` для определения цвета компонента или свойство `Caption` для установки текста на компоненте.

Вы также можете определить события, которые будут генерироваться компонентом в определенных ситуациях или действиях. Например, вы можете добавить событие `OnClick`, которое будет срабатывать при щелчке на компоненте, чтобы позволить другим частям кода реагировать на это событие.

Методы пользовательского компонента представляют действия или операции, которые можно выполнить с помощью компонента. Например, вы можете добавить метод `Draw` для рисования чего-либо на компоненте или метод `Calculate` для выполнения сложных вычислений.

При настройке пользовательских компонентов вы определяете, какие свойства, события и методы будут доступны для изменения и использования разработчиками приложений, использующих ваш компонент. Это позволяет создавать гибкие и переиспользуемые компоненты, котоможно легко настраивать и интегрировать в различные проекты.

**Функции и процедуры для работы с реестром**

Для работы с реестром существует в основном 2 категории:

1.Функции из библиотеки API

2.Функции из библиотеки Registry разработанные для Delphi

API:

где key - раздел , subkey – подраздел

если возвращаемое значение ERROR SUCCES то всё успешно

1. RegCreateKey(key:Hkey; SubKey:Pchar; var:Result:Hkey):LongInt - для создания подраздела

2. RegOpenKey(key:Hkey; Subkey:pchar; var Result:Hkey):longint – для открытия раздела

3. RegCloseKey(key:Hkey):longint – для закрытия раздела

4. RegDeleteKey(key:Hkey; Subkey:pchar) – для удаления раздела

5. RegEnumKey(key:Hkey; index:longInt: buffer:pchar; cb:longInt):longint – для получения имён всех подразделов для заданного раздела, index-счётчик на количество вызовов функции, cb – размер буфера

6. RegQueryValue(key:Hkey; SubKey:Pchar; value:pchar; var cb:longint)longint – возвращает текстовый строки связанной с ключом и подключом

7. RegSetValue(key:Hkey; SubKey:Pchar; ValueType:longint; value:pchar; cb:longint):longint – для установки нового значения ключа

TRegistry: существует отдельный класс в виде библиотеки для работы с реестром в языке делфи. Все объекты, реализуемые для работы с реестром должны объявляться этим же типом. Чтобы использовать данный класс нужно в модулях подключить Registry

Var REG: TRegistry

REG:= TRegistry.Create

REG:= TRegistry.Destroy

Свойства:

1. RootKey : HKEY - это свойство возвращает значения типа HKEY допуская чтение и запись, кроме того определяет корневой ключ реестра.

Допустимыми значениями это свойства являются: HKEY\_CLASSES\_ROOT,

HKEY\_CURRENT\_USER , HKEY\_LOCAL\_MACHINE,

HKEY\_USERS , HKEY\_CURRENT\_CONFIG

2. CurrentPath : string - возвращает путь к текущему ключу

3. LazyWrite : Boolean - "ленивая запись", по умолчанию возвращает true и оно означает что на момент возврата из функции закрытие ключа не

гарантируется что изменения в нем записаны в реестр. Если false то на момент возврата из функции закрытия ключа гарантируется что все изменения уже записаны в реестр.

CurrentKey : HKEY - возвращает открытый в настоящее время ключ реестра

Методы: (func - функция, proc - процедура)

1. proc CloseKey - для закрытия ключа реестра

2. func CreateKey(key:string):boolean; - Для создания ключа

'\.Default\Software\MyProgram' - пример значения key (до MyProgram должен быть существующий путь иначе ошибка)

3. func DeleteKey(key:string):boolean; - удаляет ключ

4. func DeleteValue(name:string):boolean; - удаляет параметр с заданным именем текущего ключа. Для этого ключ заранее должен быть открытым.

5. func GetDataInfo(valuename:string, var value:TRegDataInfo):boolean; - (2-ой параметр возвращаемый) для получения информации о параметре заданной первым аргументом текущего ключа, информация типа размер… если удалось получить true в противном случае false

TRegDataType=(rdUnknown, rdString, rdExpandString, rdInteger, rdBinary)

TRegDataInfo=record RegData: TRegDataType; запись состоит из полей

DataSize: Integer;end;

GetDataSize, GetDataType

6. func GetDataSize(valuename:string):integer; - для получения размера параметра с заданным именем текущего ключа в байтах

7. func GetDataType(valeuname:string):TRegDataType; - для получения типа текущего ключа с заданным именем данная функция возвращает одно изэтих значений: rdUnknown,rdString,rdExpandString,rdInteger,rdBinary

8. func GetKeyInfo(var valuename:TRegKeyInfo):boolean; - для получения информации о ключе. True если получили информация о ключе, в противном случае false. Значением параметра функции является заполненная следующая структура

TRegKeyInfo=record

MaxSubKeyLen:integer; // кол-во подключей

NumValues:integer; // кол-во параметров

MaxValueLen:integer; // максимальная длина имени параметра

MaxDataLen:integer; // максимальная длина данных

FileTime:TFileTime; // время последней записи ключа

end;

9. proc GetKeyNames(str:TStrings); - заполняет указанный список str списком подключей текущего ключа

10. proc GetValueNames(str:TStrings); - заполняет указанный список str списком параметров текущего ключа

11. func HasSubKeys:boolean - возвращает true если текущий ключ имеет подключи и false в противном случае

12. func KeyExists(key:string):boolean; - возвращает true если ключ key существует и false в противном случае

13. fucn LoadKey(key,filename:string):boolean; - создает ключ key и загружает в него данные из файла с именем filename, true при успехе и тд

14. proc MoveKey(oldname,newname:string,delete:boolean); - предназначена для копирования или перименования ключа. Копирует все из oldname в newname и после копирования анализируеться delete. Если его значения true то ключ oldname уничтожаеться со всем содержимым,если false то ключ старые версии oldname копируются но не уничтожаются

15. func OpenKey(key:string,cancreate:boolean):boolean; - предназначена для открытия ключа, true если удалось открыть ключ иначе false, если cancreate = true при отсутствие ключа он его создаст, а при значение false нет

16. func OpenKeyReadOnly(key:string):boolean; - открывает ключ в режиме "только для чтения". true если удалось открыть в противном случае false

READ

17. func ReadBinaryData(name:string,var buffer,buffsize:integer):integer; - для чтения данных двочных параметров с заданным именем для текущего открытого ключа в buffer размера bufsize

18. func ReadBool(name:string):boolean; - для чтения значения параметра name типа boolean(логического типа).19. func ReadDate(name:string):TDate - для чтения значения параметра name типа дата

20. func ReadDateTime(name:string):TDateTime - для чтения значения параметра name типа дата/время

21. func ReadTime(name:string):TTime - для чтения значения параметра name типа время

22. func ReadFloat(name:string):Double - для чтения значения параметра name вещественного типа

23. func ReadInteger(name:string):integer - для чтения значения параметра name целого типа

24. func ReadString(name:string):string - для чтения значения параметра name строкового типа

25. func RegistryConnect(UNCname:string):boolean; - предназначена для подключения реестра сетового компютера

UNCname ---> [\\сет\_имя\_комп](file:///\\сет_имя_комп)

26. proc RenameValue(oldanme,newname:string); - для переименования параметра текущего ключа

27. func ReplaceKey(key,filename,backupfilename:string):boolean; - для замены места хранения ключа, в backupfilename будет копия ключа

28. func RestoreKey(key,filename:string):boolean; - для открытия указанного ключа и перезаписывания его данных и подключей даннымизагруженными из файла filename

29. func SaveKey(key,filename:string):boolean; - для сохранения всех параметров указанного ключа и всех его подключей в указанном файле filename

30. func UnLoadKey(key:string):boolean; - удаляет отдельные файлы ключа из реестра

31. func ValueExists(name:string):boolean; - предназначен для проверки существования параметра с заданным именем name в текущем ключе

32. proc WriteBinaryData(name:string,var buffer, bufsize:integer);

33. proc WriteDate(name:string,date:TDate);

34. proc WriteDateTime(name:string,date:TDateTime);

35. proc WriteTime(name:string,date:TTime);

36. proc ReadFloat(name:string,data:double);

37. proc WriteInteger(name:string,data:integer);

38. proc WriteBool(name:string,data:boolean);

39. proc WriteString(name:string,data:string);

40. proc WriteExpandString(name:stringm data:TStrings);

41. proc WriteCurrency(name:string,data:currency);

**Способы использования пользовательских компонент**

В Delphi пользовательские компоненты могут быть использованы в различных сценариях для улучшения процесса разработки и создания переиспользуемых элементов интерфейса. Вот несколько способов использования пользовательских компонентов в Delphi:

1. Создание пользовательских элементов управления: Вы можете создать собственные пользовательские элементы управления (например, кнопки, поля ввода, списки и т.д.) с помощью наследования от стандартных компонентов Delphi или создания собственных компонентов с нуля.

2. Расширение функциональности существующих компонентов: Вы можете создать наследуемый класс от существующего компонента и добавить дополнительную функциональность или изменить поведение компонента по своему усмотрению.

3. Создание пользовательских диалоговых окон: Пользовательские компоненты могут быть использованы для создания собственных диалоговых окон или форм, которые могут быть повторно использованы в различных частях вашего приложения.

4. Использование библиотек компонентов: Вы можете создать библиотеку пользовательских компонентов, которую можно будет использовать в различных проектах и даже делиться с другими разработчиками.

5. Интеграция с внешними библиотеками и API: Пользовательские компоненты могут быть использованы для интеграции с внешними библиотеками и сервисами, такими как базы данных, веб-сервисы и другие API.

6. Создание пользовательских отчетов и форматирования: Пользовательские компоненты могут быть использованы для создания пользовательских отчетов, форматирования данных и представления информации в удобном для пользователя виде.

7. Создание специализированных компонентов для конкретной отрасли: Вы можете создать пользовательские компоненты, специально адаптированные под определенные отрасли, такие как медицинская, финансовая, научная и т.д. Это позволит ускорить разработку приложений для этих отраслей.

8. Разработка пользовательских анимаций и визуальных эффектов: Пользовательские компоненты могут быть использованы для создания красивых анимаций, переходов и визуальных эффектов, которые делают пользовательский интерфейс более привлекательным и интерактивным.

9. Использование пользовательских компонентов в многопользовательских приложениях: Если вы разрабатываете приложение с множеством пользователей, то пользовательские компоненты могут помочь в создании персонализированных интерфейсов для различных категорий пользователей.

10. Создание плагинов и расширений: Пользовательские компоненты могут быть использованы для создания плагинов и расширений, которые позволяют динамически расширять функциональность вашего приложения без изменения его основного кода.

11. Использование пользовательских компонентов в тестировании: Вы можете создать пользовательские компоненты для автоматизации тестирования вашего приложения, что поможет упростить процесс тестирования и обеспечить более высокое качество вашего приложения.

12. Создание пользовательских компонентов для мобильных приложений: Если вы разрабатываете мобильные приложения с использованием Delphi, пользовательские компоненты могут быть использованы для создания пере используемых элементов интерфейса для iOS и Android.

**Способы создания пользовательских функций(Excel)**

Хотя в Excel предлагается большое число встроенных функций, в нем может не быть той функции, которая нужна для ваших вычислений. К сожалению, разработчики Excel не могли предугадать все потребности пользователей. Однако в Excel можно создавать собственные функции, и ниже вы найдете все нужные для этого инструкции.

Создание простой пользовательской функции

Пользовательские функции (как и макросы) записываются на языке программирования **Visual Basic для приложений (VBA)**. Они отличаются от макросов двумя вещами. Во-первых, в них используются процедуры *Function*, а не *Sub*. Это значит, что они начинаются с оператора **Function**, а не **Sub**, и заканчиваются оператором **End Function**, а не **End Sub**. Во-вторых, они выполняют различные вычисления, а не действия. Некоторые операторы (например, предназначенные для выбора и форматирования диапазонов) исключаются из пользовательских функций. Из этой статьи вы узнаете, как создавать и применять пользовательские функции. Для создания функций и макросов используется **редактор Visual Basic (VBE)**, который открывается в отдельном окне.

Правила создания пользовательских функций

Пользовательские функции должны начинаться с оператора Function и заканчиваться оператором End Function. Помимо названия функции, оператор Function обычно включает один или несколько аргументов. Однако вы можете создать функцию без аргументов. В Excel доступно несколько встроенных функций (например, СЛЧИС и ТДАТА), в которых нет аргументов.

После оператора Function указывается один или несколько операторов VBA, которые проверят соответствия условиям и выполняют вычисления с использованием аргументов, переданных функции. Наконец, в процедуру функции следует включить оператор, назначающий значение переменной с тем же именем, что у функции. Это значение возвращается в формулу, которая вызывает функцию.

Применение ключевых слов VBA в пользовательских функциях

Количество ключевых слов VBA, которые можно использовать в пользовательских функциях, меньше числа, используемого в макросах. Настраиваемые функции не могут выполнять другие задачи, кроме возврата значения в формулу на этом или в выражение, используемом в другом макросе или функции VBA. Например, пользовательские функции не могут изменять размер окна, редактировать формулу в ячейке, а также изменять шрифт, цвет или узор текста в ячейке. Если в процедуру функции включить такой код действия, функция возвращает #VALUE! ошибку "#ВЫЧИС!".

Единственное действие, которое может выполнять процедура функции (кроме вычислений), — это отображение диалогового окна. Чтобы получить значение от пользователя, выполняющего функцию, можно использовать в ней оператор **InputBox**. Кроме того, с помощью оператора **MsgBox** можно выводить сведения для пользователей. Вы также можете использовать настраиваемые диалоговые окна (**UserForms**), но эта тема выходит за рамки данной статьи.

Создание пользовательских функций в Excel позволяет вам расширить функциональность программы, добавив собственные пользовательские функции, которые могут быть использованы так же, как встроенные функции Excel. Вот как создать пользовательскую функцию в Excel:

1. Откройте Excel и перейдите во вкладку "Разработчик". Если у вас нет вкладки "Разработчик", то ее можно активировать, выбрав "Файл" -> "Параметры" -> "Настройка ленты" и поставив галочку напротив "Разработчик".

2. Нажмите на кнопку "Визуальный редактор" на вкладке "Разработчик", чтобы открыть редактор VBA (Visual Basic for Applications).

3. В редакторе VBA выберите "Вставка" -> "Модуль", чтобы добавить новый модуль.

4. Введите код для вашей пользовательской функции. Например, если вы хотите создать пользовательскую функцию для сложения двух чисел, код может выглядеть следующим образом:

Function MyCustomSum(num1 As Double, num2 As Double) As Double

MyCustomSum = num1 + num2

End Function

5. Сохраните модуль, нажав "Файл" -> "Сохранить" или используя комбинацию клавиш "Ctrl + S".

6.Теперь ваша пользовательская функция доступна для использования в любой ячейке Excel. Просто введите "=MyCustomSum(A1, B1)", где A1 и B1 - это ячейки с числами, которые вы хотите сложить.

Для макросов:

Private Sub CommandButton1\_Click()

Dim maxVal As Double

Dim minVal As Double

Dim avgVal As Double

maxVal = getmaxvalue(Worksheets("Лист2").range("B5:B28"))

minVal = getminvalue(Worksheets("Лист2").range("B5:B28"))

avgVal = getaveragevalue(Worksheets("Лист2").range("B5:B28"))

Label1.Caption = maxVal

Label2.Caption = minVal

Label3.Caption = avgVal

Label4.Caption = Worksheets("Лист2").range("B29").Value

Label1.BackColor = RGB(0, 255, 0)

Label1.Font.Size = 12

Label2.BackColor = RGB(255, 0, 0)

Label2.Font.Size = 12

Label3.BackColor = RGB(255, 255, 0)

Label3.Font.Name = "Arial"

Label3.Font.Size = 12

Label4.BackColor = RGB(0, 255, 255)

Label4.Font.Size = 12

Пример пользовательских функций:

Function getmaxvalue(range As range)

Dim i As Double

For Each cell In range.cells

If cell.Value > i Then

i = cell.Value

End If

Next

getmaxvalue = i

End Function

Function getminvalue(range As range) As Double

Dim min As Double

min = range.cells(1).Value

For Each cell In range.cells

If cell.Value < min Then

min = cell.Value

End If

Next cell

getminvalue = min

End Function

Function getaveragevalue(range As range)

Dim sum As Double

Dim count As Integer

sum = 0

count = 0

For Each cell In range

sum = sum + cell.Value

count = count + 1

Next cell

If count > 0 Then

getaveragevalue = sum / count

Else

getaveragevalue = 0

End If

End Function

**Способы использования пользовательских функций в excel**

Пользовательские функции в Excel могут быть использованы для различных целей, включая расчеты, обработку данных, конвертацию единиц измерения и многое другое. Вот несколько способов использования пользовательских функций в Excel:

1. **Расчеты и формулы**: Пользовательские функции могут быть использованы для выполнения сложных математических операций, обработки данных и создания новых формул. Например, вы можете создать пользовательскую функцию для расчета среднего значения, медианы, стандартного отклонения и т.д.

2. **Обработка текста**: Пользовательские функции могут помочь в обработке текстовых данных, таких как удаление лишних пробелов, форматирование текста, извлечение подстрок и т.д.

3. **Конвертация единиц измерения**: Если вам часто приходится конвертировать единицы измерения (например, дюймы в сантиметры, фунты в килограммы и т.д.), вы можете создать пользовательскую функцию для автоматизации этого процесса.

4. **Поиск и фильтрация данных**: Пользовательские функции могут быть использованы для поиска и фильтрации данных в таблицах. Например, вы можете создать функцию для поиска определенного значения в столбце или для фильтрации данных по определенным критериям.

5. **Автоматизация задач**: Пользовательские функции могут помочь автоматизировать выполнение повторяющихся задач, таких как расчеты, анализ данных и создание отчетов.

6. **Интеграция с внешними источниками данных**: Если у вас есть нестандартный источник данных (например, API, база данных и т.д.), вы можете создать пользовательскую функцию для получения и обработки этих данных непосредственно в Excel.

7. **Генерация случайных данных**: Пользовательские функции могут быть использованы для генерации случайных чисел, текстовых значений или даже случайных дат.

8. **Работа с датами и временем**: Вы можете создать пользовательскую функцию для выполнения различных операций с датами и временем, таких как вычисление разницы между двумя датами, добавление определенного количества дней к дате и т.д.

9. **Анализ данных**: Пользовательские функции могут помочь в анализе больших объемов данных, выполнении расчетов, создании сводных таблиц и графиков.

10. **Работа с массивами данных**: Если вам нужно обрабатывать массивы данных, вы можете создать пользовательскую функцию для выполнения операций над массивами, таких как сортировка, фильтрация, поиск значений и т.д.

11. **Создание пользовательских отчетов**: Пользовательские функции могут быть использованы для создания индивидуальных отчетов и дашбордов на основе данных из вашей таблицы Excel.

12. **Интеграция с другими приложениями Microsoft Office**: Вы можете использовать пользовательские функции для интеграции данных из Excel с другими приложениями Microsoft Office, такими как Word, PowerPoint и Outlook.

...

**Способы создания умных и сводных таблиц(Excel)**

Умные или динамические таблицы в excel это таблица, оформляемая специальным образом, который поддерживает выполнения различных операций в автоматическом виде и поддержкой некоторых других дополнительных функциональности. С динамическими или умными таблицами можно выполнять операции фильтрации, поиска и подбора данных с различными критериями кроме того вставки срезов, операций экспорта других таблиц создание сводных таблиц и создание динамических графиков. Для того чтобы обычную таблицу преобразовать в умную существует несколько способов. Прейти в ячейки с данными:

1. Воспользоваться различными видами форматирования таблиц на вкладке главная (форматирование таблиц)
2. Воспользоваться элементом управления таблиц на вкладке вставка (вставка таблица)
3. Нажатие на сочетание клавиш Ctrl + T

После выполнения одного из перечисленных действий появляется диалоговое окно указания диапазона данных и строки заголовка после настройки таблицы автоматически преобразуется в умную или добавление новых значений в столбце или строке следующий за строками или столбцами умной таблицы весь его диапазон добавляется в умную таблицу

Для выполнения перечисленных операций в умных таблицах не требуется никаких дополнительных функций или процедур(матрикса)

**Способы создания динамических отчётов**

Динамический отчет в Excel - это отчет, который автоматически обновляется и адаптируется к изменениям во входных данных. Он позволяет пользователям быстро и легко получать актуальную информацию на основе обновленных данных, без необходимости вручную изменять структуру или содержание отчета.

Динамический отчет может включать в себя различные элементы, такие как сводные таблицы, графики, сводные диаграммы, условное форматирование и другие элементы визуализации данных. Он может быть настроен таким образом, чтобы автоматически обновляться при добавлении новых данных, изменении параметров фильтрации или сортировки, или при изменении других параметров.

Преимущества динамических отчетов включают удобство использования, возможность быстро получать актуальную информацию, возможность быстро анализировать данные и принимать решения на основе обновленной информации.

Создание динамического отчета в Excel обычно включает использование функций, формул, сводных таблиц, графиков и других инструментов для автоматизации процесса обновления и адаптации отчета к изменениям в данных.

Существует несколько способов создания динамических отчетов в Excel. Вот некоторые из них:

1. Использование сводных таблиц (Pivot Tables): Сводные таблицы позволяют быстро и легко анализировать большие объемы данных и создавать динамические отчеты. Вы можете изменять структуру сводной таблицы, переупорядочивать данные, добавлять новые поля и фильтры для динамического анализа.
2. Использование макросов (VBA): Если вам нужно создать сложные динамические отчеты с большим количеством автоматизации, вы можете использовать язык программирования VBA (Visual Basic for Applications), встроенный в Excel. С помощью VBA вы можете создавать пользовательские функции, макросы и формы для управления данными и создания динамических отчетов.
3. Использование сводных графиков (Pivot Charts): Подобно сводным таблицам, сводные графики позволяют визуализировать данные и создавать динамические отчеты на основе изменения структуры данных.
4. Использование таблиц (Tables): В Excel можно создавать таблицы, которые автоматически расширяются или сжимаются при добавлении или удалении данных. Это позволяет создавать динамические отчеты, которые автоматически адаптируются к изменениям в данных.

5. Использование динамических диапазонов и именованных диапазонов: Вы можете создавать динамические диапазоны с помощью формул или функций, таких как СМЕЩ(INDIRECT), чтобы автоматически расширять или сжимать диапазон в зависимости от изменений в данных. Также можно использовать именованные диапазоны для удобного обращения к данным в формулах и отчетах.

**Способы нестандартных диаграммы (Excel)**

Диаграмма - это визуальное представление данных, которое позволяет легче воспринимать и анализировать информацию. Диаграммы могут быть использованы для отображения различных видов данных, таких как числовые значения, пропорции, отношения между различными категориями и т.д.

Существует множество различных типов диаграмм, каждый из которых подходит для определенных видов данных и целей анализа. Некоторые из наиболее распространенных типов диаграмм включают столбчатые диаграммы, круговые диаграммы, линейные графики, диаграммы рассеяния и т.д.

Диаграммы обычно создаются с использованием специальных программ, таких как Microsoft Excel, Google Sheets, Tableau, Power BI и другие. Создание диаграмм позволяет наглядно представить данные и делать выводы на основе их визуального анализа.

Создание нестандартных диаграмм в Excel может быть достигнуто различными способами, включая использование стандартных инструментов Excel и дополнительных приемов. Вот несколько способов создания нестандартных диаграмм:

1. Использование комбинированных диаграмм: В Excel можно создавать комбинированные диаграммы, объединяя различные типы диаграмм, такие как столбчатые, линейные, круговые и другие. Это позволяет отобразить несколько наборов данных на одной диаграмме для более полного анализа.

2. Использование условного форматирования: Вы можете использовать условное форматирование, чтобы изменять цвет или стиль элементов диаграммы в зависимости от значений данных. Например, вы можете изменить цвет столбцов на гистограмме в зависимости от их значения.

3. Использование пользовательских шаблонов: Вы можете создать собственные пользовательские шаблоны для диаграмм, чтобы изменить их внешний вид и стиль, включая цвета, шрифты, размеры и расположение элементов.

4. Использование дополнительных инструментов: Вы также можете использовать дополнительные инструменты и плагины, доступные для Excel, для создания более сложных и нестандартных типов диаграмм, таких как радарные диаграммы, диаграммы Ганта, тепловые карты и другие.

5. Использование макросов: Если вам нужно создать очень специфический тип диаграммы, который недоступен в стандартных инструментах Excel, вы можете использовать макросы VBA (Visual Basic for Applications) для создания пользовательских функций и процедур, которые будут генерировать необходимую диаграмму.

7. Создание диаграммы с использованием дополнительных элементов управления: Вы можете добавлять дополнительные элементы управления, такие как кнопки, флажки или список раскрывающихся меню, чтобы позволить пользователям выбирать, какие данные отображать на диаграмме.

8. Использование диаграммы гистограммы с накоплением: Диаграмма гистограммы с накоплением позволяет отображать, как общая сумма наблюдений накапливается по категориям. Это может быть полезно для анализа распределения данных.

9. Использование 3D-диаграмм: Excel позволяет создавать 3D-диаграммы, которые могут придать вашим данным более объемный вид и помочь визуализировать сложные трехмерные данные.

10. Создание пользовательских диаграмм с использованием программирования: Если у вас есть опыт в программировании, вы можете использовать язык программирования VBA для создания пользовательских диаграмм, которые полностью соответствуют вашим требованиям и спецификациям.

**Способы создание диаграммы – карты**

В Microsoft Excel существует несколько способов создания диаграммы-карты. Вот несколько шагов, которые помогут вам создать диаграмму-карту в Excel:

1. Подготовьте данные: У вас должны быть данные, связанные с географическими областями (например, страны, регионы, штаты), и числовые значения, которые вы хотите отобразить на карте.

2. Вставка диаграммы-карты:

- Выберите ваши данные.

- Перейдите на вкладку "Вставка" в меню Excel.

- Нажмите на кнопку "Карты" в группе "Вставка диаграмм".

3. Выбор типа карты: В появившемся меню выберите тип карты, который соответствует вашим данным (например, "Точечная-пузырьковая").

4. Настройка параметров карты:

- После создания карты можно настроить параметры отображения данных,

убрать название диаграммы, легенду, удаляем все данные и добавляем новые из умной таблицы,

удаляем линии сетки и значение х,у.

нажимаем двойной щелчок мыши и выбираем формат области построения, заливка - рисунок текстура – файл выбираем файл нашей карты

5. Сохранение и настройка:

- После создания и настройки карты сохраните свою работу.