PureBasic с нуля до архитектуры

Полное учебное издание

Автор: Роман Бурмистров
Язык: PureBasic
Издание: Учебное руководство
Версия: 1.0

Аннотация

Данное издание представляет собой последовательный курс по изучению языка PureBasic — от основ программирования до построения архитектурных решений и создания собственных GUI-фреймворков.

Книга ориентирована на начинающих разработчиков и постепенно выводит читателя на профессиональный уровень.

Содержание

Часть I. Основы программирования

1. Что такое программирование
2. Язык PureBasic
3. Установка среды разработки
4. Первая программа
5. Переменные
6. Типы данных и арифметика
7. Условные конструкции
8. Циклы
9. Процедуры

Часть II. Работа с данными

10. Строки и работа с текстом
11. Массивы
12. Списки
13. Структуры
14. Мар (словари)
15. Работа с файлами

Часть III. Создание графических приложений

16. Создание окна

17. Элементы управления (Gadgets)

18. Обработка событий

19. Меню и диалоговые окна

20. Мини-проект: Калькулятор

Часть IV. Практическая разработка

21. Итоговый проект: Менеджер заметок

Часть V. Продвинутый уровень

22. Модули

23. Работа с памятью

24. Создание собственных библиотек

25. Работа с Windows API

26. Создание собственного GUI-фреймворка

Введение

Программирование — это процесс создания инструкций для компьютера.
Язык PureBasic выбран в качестве инструмента обучения благодаря своей простоте, скорости и возможностям системного уровня.

Цель книги — сформировать:

- фундаментальные знания;
- практические навыки;
- понимание архитектуры приложений;
- способность к самостоятельной разработке.

Структура глав

Каждая глава содержит:

1. Теоретическую часть
2. Примеры кода
3. Разбор примеров
4. Типичные ошибки
5. Практические задания

Обозначения в книге

- `код` — фрагменты программы
- .i, .s, .d — типы данных
- #Константа — системные константы PureBasic
- :: — обращение к модулю

Методические рекомендации

Для максимальной эффективности:

1. Набирать примеры вручную.
2. Экспериментировать с кодом.

3. Выполнять все практические задания.
4. Создать собственный проект после завершения книги.

Глава 1

Что такое программирование

1.1 Что вообще делает программист?

Компьютер сам по себе ничего не «понимает».
Он выполняет инструкции.

Программа — это список команд, которые мы даём компьютеру.

Пример из жизни:

Рецепт — это программа для повара.
Инструкция по сборке — это программа для человека.

Точно так же код — это инструкция для компьютера.

1.2 Что такое язык программирования?

Компьютер понимает только нули и единицы.

Но писать так неудобно. Поэтому были придуманы языки программирования — специальные языки для общения с компьютером.

Примеры языков:

- Python
- C++
- Java
- Basic
- PureBasic

В этой книге мы будем изучать **PureBasic** — простой и мощный язык для создания программ.

1.3 Почему PureBasic — хороший выбор для новичка?

PureBasic:

- Очень простой синтаксис
- Минимум «лишней магии»
- Позволяет создавать .exe файлы
- Подходит для Windows, Linux и macOS
- Быстро компилируется

Это отличный язык, чтобы понять основы программирования.

1.4 Как работает программа?

Программа проходит 3 этапа:

1. Вы пишете код
2. Компилятор переводит его в машинный код
3. Компьютер выполняет программу

PureBasic — компилируемый язык.
Это значит, что на выходе получается настоящий .exe файл.

Глава 2

Язык программирования PureBasic

2.1 Общая характеристика языка

PureBasic — это компилируемый язык программирования высокого уровня, предназначенный для создания нативных приложений.

Язык ориентирован на:

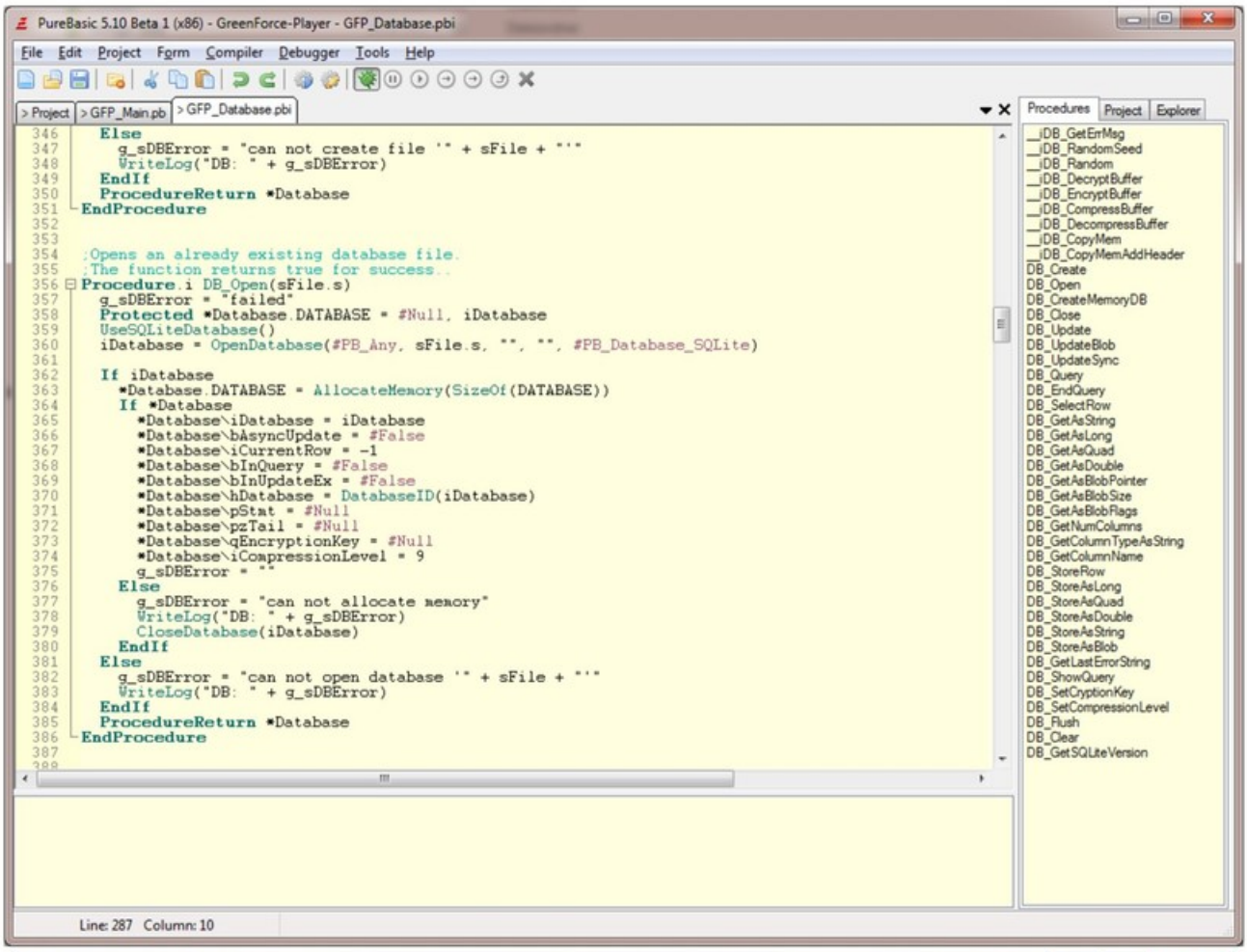
- простоту синтаксиса;
- высокую скорость компиляции;
- создание самостоятельных исполняемых файлов;
- кроссплатформенную разработку.

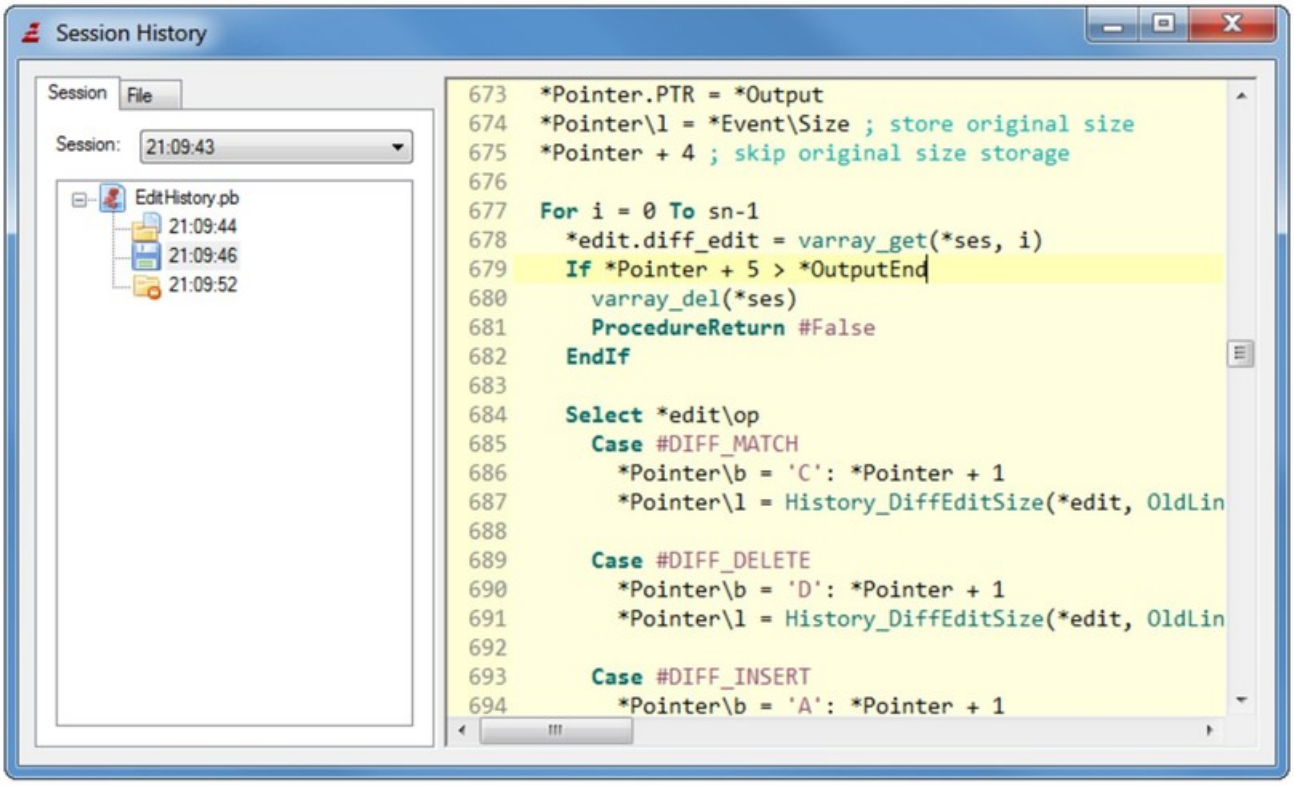
PureBasic поддерживает создание программ для операционных систем:

- Windows
- Linux
- macOS

После компиляции создаётся автономный исполняемый файл, не требующий дополнительных библиотек или виртуальных машин.

2.2 История и назначение языка





4

Язык **PureBasic** был создан как современная реализация идей классического BASIC с сохранением простоты и добавлением возможностей низкоуровневого программирования.

Основные задачи языка:

1. Быстрая разработка приложений.
2. Создание компактных и быстрых программ.
3. Доступ к системным функциям операционной системы.
4. Разработка графических интерфейсов без сложных фреймворков.

PureBasic сочетает простоту BASIC-подобного синтаксиса и возможности работы с памятью, структурами и системными API.

2.3 Особенности PureBasic

1. Компилируемый язык

В отличие от интерпретируемых языков, программа полностью переводится в машинный код до запуска. Это обеспечивает:

- высокую скорость работы;
- отсутствие зависимости от сторонних интерпретаторов;
- компактные исполняемые файлы.

2. Простота синтаксиса

Пример программы на PureBasic:

```
MessageRequester("Сообщение", "Здравствуйте, мир!")
```

Данный код создаёт окно с текстом «Здравствуйте, мир!».

3. Нативный графический интерфейс

PureBasic использует системные элементы управления операционной системы, благодаря чему программы выглядят естественно для выбранной платформы.

4. Работа с низким уровнем

Язык поддерживает:

- указатели;
- структуры;
- работу с памятью;
- использование системных библиотек.

Это делает его подходящим как для обучения, так и для практической разработки.

2.4 Области применения

PureBasic может использоваться для:

- создания утилит;
- разработки небольших настольных приложений;
- написания инструментов автоматизации;
- обучения основам программирования;
- создания графических программ;
- разработки вспомогательных системных инструментов.

2.5 Преимущества для начинающих

Для изучающих программирование с нуля язык полезен по следующим причинам:

1. Минимальный порог входа.
2. Отсутствие сложных концепций на начальном этапе.
3. Наглядный результат работы программы.
4. Быстрое получение исполняемого файла.

Контрольные вопросы

1. К какому типу языков относится PureBasic?
2. Что означает термин «компилируемый язык»?

3. Какие операционные системы поддерживает PureBasic?

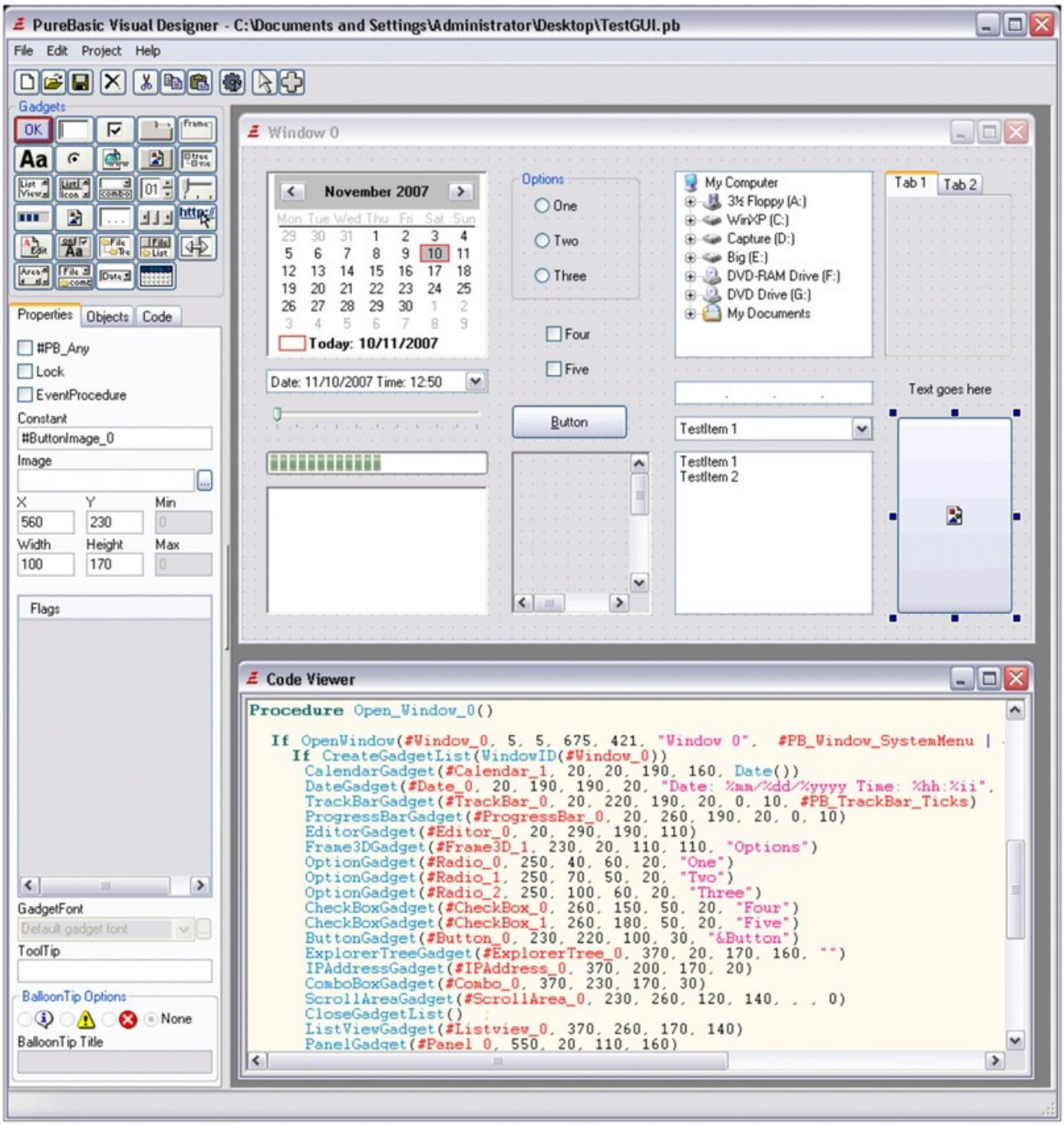
4. В чём отличие компилируемого языка от интерпретируемого?

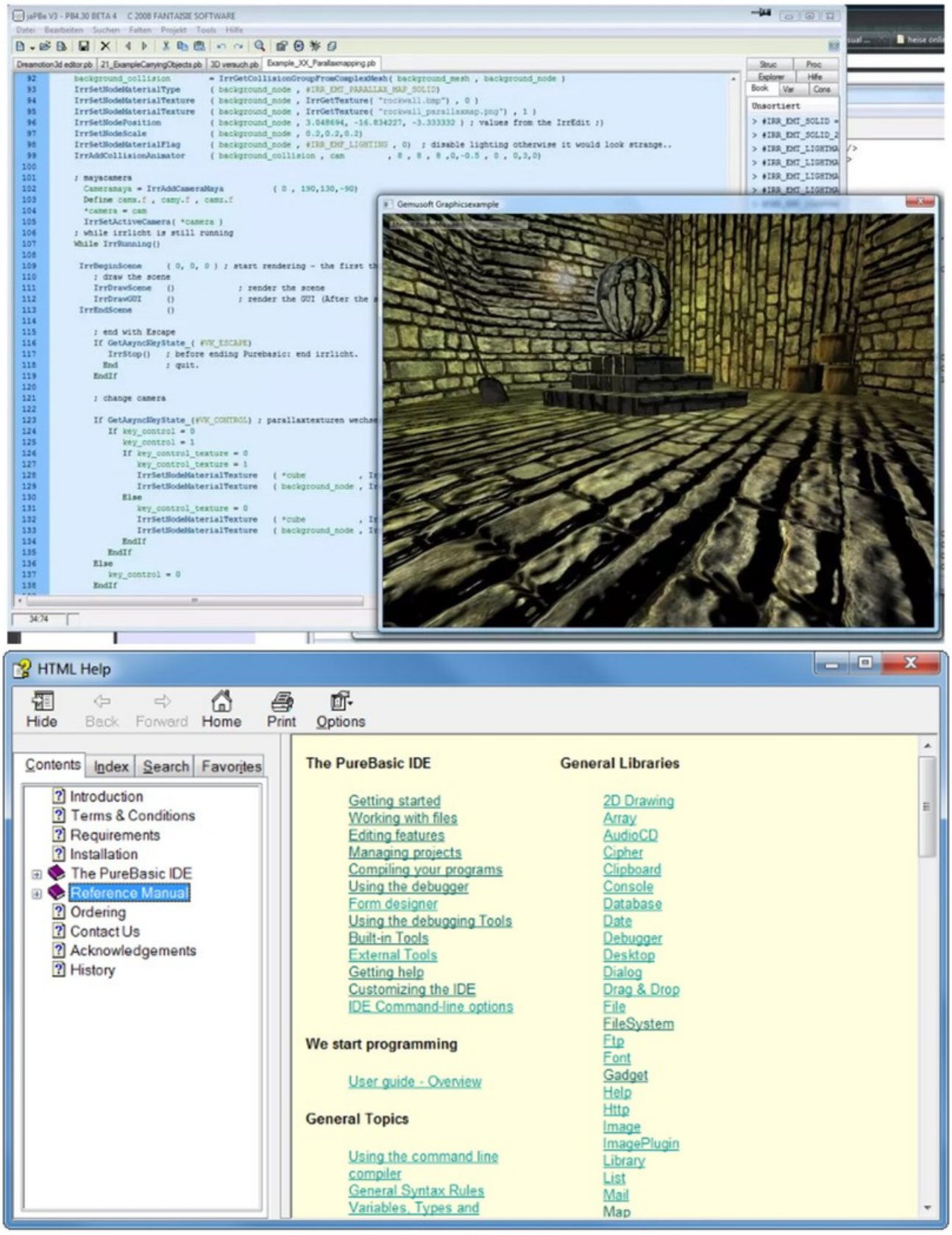
В следующей главе будет рассмотрена установка среды разработки и создание первой программы.

Глава 3

Установка PureBasic и настройка среды разработки

3.1 Получение дистрибутива


PureBasic Visual Designer - C:\Documents and Settings\Administrator\Desktop\TestGUI.pb
File Edit Project Help
Gadgets
Properties Objects Code
#PB_Any
Lock
EventProcedure
Constant
#ButtonImage_0
Image
X Y Min
560 230 0
Width Height Max
100 170 0
Flags
GadgetFont
Default gadget font
ToolTip
BalloonTip Options
None
BalloonTip Title
Window 0
November 2007
Today: 10/11/2007
Date: 11/10/2007 Time: 12:50
Options
One
Two
Three
Four
Five
Button
My Computer
3½ Floppy (A:)
WinXP (C:)
Capture (D:)
Big (E:)
DVD-RAM Drive (F:)
DVD Drive (G:)
My Documents
Tab 1
Tab 2
Text goes here
TestItem 1
TestItem 1
TestItem 2
Code Viewer
Procedure Open_Window_0()
If OpenWindow(#Window_0, 5, 5, 675, 421, "Window 0", #PB_Window_SystemMenu |
If CreateGadgetList(WindowID(#Window_0))
CalendarGadget(#Calendar_1, 20, 20, 190, 160, Date())
DateGadget(#Date_0, 20, 190, 190, 20, "Date: %mm/%dd/%yyyy Time: %hh:%ii",
TrackBarGadget(#TrackBar_0, 20, 220, 190, 20, 0, 10, #PB_TrackBar_Ticks)
ProgressBarGadget(#ProgressBar_0, 20, 260, 190, 20, 0, 10)
EditorGadget(#Editor_0, 20, 290, 190, 110)
Frame3DGadget(#Frame3D_1, 230, 20, 110, 110, "Options")
OptionGadget(#Radio_0, 250, 40, 60, 20, "One")
OptionGadget(#Radio_1, 250, 70, 50, 20, "Two")
OptionGadget(#Radio_2, 250, 100, 60, 20, "Three")
CheckBoxGadget(#CheckBox_0, 260, 150, 50, 20, "Four")
CheckBoxGadget(#CheckBox_1, 260, 180, 50, 20, "Five")
ButtonGadget(#Button_0, 230, 220, 100, 30, "&Button")
ExplorerTreeGadget(#ExplorerTree_0, 370, 20, 170, 160, "")
IPAddressGadget(#IPAddress_0, 370, 200, 170, 20)
ComboBoxGadget(#Combo_0, 370, 230, 170, 30)
ScrollAreaGadget(#ScrollArea_0, 230, 260, 120, 140, , , 0)
CloseGadgetList()
ListViewGadget(#Listview_0, 370, 260, 170, 140)
PanelGadget(#Panel_0, 550, 20, 110, 160)



4

Для начала работы необходимо загрузить установочный пакет языка **PureBasic** с официального сайта разработчиков.

На странице загрузки доступны версии для:

- Windows
- Linux
- macOS

Следует выбрать версию, соответствующую установленной операционной системе.

3.2 Установка в Windows

1. Запустите загруженный установочный файл.
2. Примите лицензионное соглашение.
3. Выберите папку установки.
4. Дождитесь завершения установки.
5. Запустите среду разработки.

После установки создаётся ярлык для запуска IDE.

3.3 Установка в Linux

1. Загрузите архив с дистрибутивом.
2. Распакуйте его в выбранную директорию.
3. Перейдите в папку программы.
4. Запустите исполняемый файл purebasic.

В большинстве случаев установка дополнительных компонентов не требуется.

3.4 Установка в macOS

1. Загрузите версию для macOS.
2. Откройте установочный пакет.
3. Переместите приложение в папку «Программы».
4. Запустите IDE.

3.5 Интерфейс среды разработки (IDE)

После запуска открывается интегрированная среда разработки (IDE).

Основные элементы интерфейса:

1. **Редактор кода** — область для написания программы.
2. **Панель инструментов** — кнопки компиляции и запуска.
3. **Строка состояния** — информация о состоянии проекта.
4. **Окно сообщений компилятора** — вывод ошибок и предупреждений.

3.6 Первая проверка работы среды

Создайте новый файл:

File → New

Введите следующий код:

```
MessageRequester("Проверка", "PureBasic установлен успешно!")
```

Нажмите кнопку **Compile / Run** (или клавишу F5).

Если появилось окно с сообщением, значит установка выполнена корректно.

3.7 Организация рабочей папки

Рекомендуется:

- создать отдельную папку для учебных проектов;
- сохранять каждую программу в отдельный файл;
- использовать понятные имена файлов.

Пример:

```
Lesson01.pb
Lesson02.pb
Calculator.pb
```

Контрольные вопросы

1. Где необходимо загружать PureBasic?
2. Что такое IDE?
3. Какая клавиша используется для запуска программы?
4. Что делать, если программа не запускается?

В следующей главе будет рассмотрена структура первой программы и принцип её работы.

Глава 4

Первая программа и её структура

4.1 Программа «Здравствуйте, мир»

Традиционно изучение любого языка программирования начинается с создания простой программы, выводящей текстовое сообщение.

Создадим следующий пример:

```
MessageRequester("Приветствие", "Здравствуйте, мир!»)
```

После компиляции и запуска на экране появится окно с указанным текстом.

4.2 Разбор программы

Рассмотрим структуру данной строки:


```
MessageRequester("Приветствие", "Здравствуйте, мир!»)
```

1. MessageRequester

Это встроенная процедура языка PureBasic.

Процедура — это готовая команда, выполняющая определённое действие.
В данном случае она создаёт диалоговое окно с сообщением.

2. Параметры процедуры

В круглых скобках передаются значения — параметры.

В нашем примере:

- Первый параметр — заголовок окна.
- Второй параметр — текст сообщения.

Параметры передаются в строгом порядке.

3. Строки

Текст в кавычках называется строкой.

Пример строки:

"Это строка текста"
Строка всегда заключается в двойные кавычки.

4.3 Пошаговый алгоритм работы программы

1. Компилятор переводит код в машинные инструкции.
2. Программа запускается.
3. Вызывается процедура MessageRequester.
4. Операционная система создаёт окно.
5. Пользователь закрывает окно.
6. Программа завершает работу.

4.4 Добавление нескольких команд

Программа может содержать несколько строк:

```
MessageRequester("Сообщение 1", "Первая строка")
MessageRequester("Сообщение 2", "Вторая строка")
```

Команды выполняются сверху вниз, последовательно.

4.5 Завершение программы

В PureBasic программа завершается автоматически после выполнения последней инструкции.

Дополнительная команда завершения не требуется.

4.6 Основные правила оформления кода

1. Одна команда — одна строка.
2. Строки текста обязательно заключаются в кавычки.
3. Важно соблюдать правильный синтаксис (скобки, запятые).

4.7 Типичные ошибки начинающих

1. Отсутствие кавычек:

   MessageRequester(Приветствие, Здравствуйте)

2. Ошибка: текст не заключён в кавычки.
3. Пропущенная закрывающая скобка:

   MessageRequester("Текст", "Ошибка"

4. Лишняя запятая.

При наличии ошибки компилятор укажет номер строки и описание проблемы.

Практическое задание

1. Создайте программу с тремя сообщениями.
2. Измените заголовки окон.
3. Попробуйте намеренно допустить ошибку и проанализировать сообщение компилятора.

Итоги главы

В данной главе были рассмотрены:

- создание первой программы;
- вызов встроенной процедуры;
- передача параметров;
- структура простейшего кода.

В следующей главе будет рассмотрено одно из ключевых понятий программирования — переменные.

Глава 5

Переменные

5.1 Понятие переменной

Переменная — это именованная область памяти, предназначенная для хранения данных.

Иными словами, переменная позволяет сохранить значение для последующего использования в программе.

Пример из повседневной жизни:
Коробка с подписью — это переменная.
Содержимое коробки — это значение переменной.

5.2 Зачем нужны переменные

Без переменных программа может выполнять только фиксированные действия.
Переменные позволяют:

- хранить числа;
- хранить текст;
- выполнять вычисления;
- изменять данные во время работы программы.

5.3 Объявление переменных в PureBasic

В языке **PureBasic** имя переменной обычно начинается со специального символа, обозначающего тип данных.

Простейший пример:

```
number = 10
```

В данном случае создаётся переменная number, которой присваивается значение 10.

5.4 Типы переменных (введение)

В PureBasic используются разные типы данных. На начальном этапе рассмотрим два основных типа:

1. Целые числа

Обозначаются суффиксом .i

```
age.i = 25
```

2. Строки

Обозначаются суффиксом .s

```
name.s = "Алексей"
```

5.5 Вывод значения переменной

Переменную можно использовать внутри процедур.

Пример:

```
name.s = "Мария"
MessageRequester("Имя пользователя", name)
```

В данном случае в окне будет выведено содержимое переменной.

5.6 Изменение значения переменной

Значение переменной можно изменять:

counter.i = 1
counter = 2
counter = 3
Переменная хранит только текущее значение.

5.7 Использование переменных в вычислениях

```
a.i = 10
b.i = 5
sum.i = a + b

MessageRequester("Результат", Str(sum))
```

Обратите внимание

Для вывода числа используется функция Str(), которая преобразует число в строку.

5.8 Правила именования переменных

1. Имя должно начинаться с буквы.
2. Нельзя использовать пробелы.
3. Рекомендуется использовать осмысленные названия:
    - age
    - totalSum
    - userName

5.9 Типичные ошибки

1. Попытка использовать переменную без присвоения значения.
2. Ошибка в суффиксе типа.
3. Попытка вывести число без преобразования в строку.

Пример ошибки:

```
MessageRequester("Ошибка", sum)
```

Правильно:

```
MessageRequester("Ошибка", Str(sum))
```

Практическое задание

1. Создайте переменную с вашим возрастом.
2. Создайте переменную с вашим именем.
3. Выведите сообщение вида:

   Меня зовут <имя>, мне <возраст> лет.

Подсказка: используйте Str() для преобразования числа.

Итоги главы

В данной главе были рассмотрены:

- понятие переменной;
- объявление переменных;
- основные типы данных;

- изменение значений;
- вывод переменных.

Глава 6

Типы данных и арифметические операции

6.1 Понятие типа данных

Тип данных определяет:

- какие значения может хранить переменная;
- сколько памяти она занимает;
- какие операции можно выполнять с её значением.

В языке **PureBasic** тип переменной указывается с помощью суффикса после имени.

6.2 Основные числовые типы

На начальном этапе рассмотрим наиболее часто используемые типы.

1. Целое число — .i

Используется для хранения целых чисел:

number.i = 100
Диапазон зависит от разрядности системы (32 или 64 бита).

2. Дробное число — .f и .d

.f — число с плавающей точкой (одинарная точность)

price.f = 10.5
.d — число с двойной точностью

pi.d = 3.1415926535
Тип .d обеспечивает большую точность вычислений.

3. Строка — .s

Используется для хранения текста:

text.s = "Пример строки"

6.3 Арифметические операции

PureBasic поддерживает стандартные математические операторы.

Оператор	Описание	Пример
+	Сложение	a + b
–	Вычитание	a - b
*	Умножение	a * b
/	Деление	a / b

6.4 Пример вычислений

```
a.i = 20
b.i = 5

sum.i = a + b
difference.i = a - b
product.i = a * b
quotient.i = a / b

MessageRequester("Результат",
  "Сумма: " + Str(sum) + #CRLF$ +
  "Разность: " + Str(difference) + #CRLF$ +
  "Произведение: " + Str(product) + #CRLF$ +
  "Частное: " + Str(quotient))
```

Разбор:

- Str() преобразует число в строку.
- + используется для объединения строк.
- #CRLF$ добавляет перенос строки.

6.5 Деление целых чисел

Важно учитывать, что при делении целых чисел результат также будет целым числом.

a.i = 7
b.i = 2
result.i = a / b ; результат будет 3
Для получения дробного результата необходимо использовать тип .f или .d:

a.d = 7
b.d = 2
result.d = a / b ; результат будет 3.5

6.6 Порядок выполнения операций

PureBasic соблюдает стандартный математический порядок:

1. Скобки
2. Умножение и деление
3. Сложение и вычитание

Пример:

```
result.i = 2 + 3 * 4    ; результат будет 14
```

Использование скобок:

```
result.i = (2 + 3) * 4  ; результат будет 20
```

6.7 Инкремент и декремент

Увеличение значения на единицу:

counter.i = 5
counter + 1
Уменьшение значения:

counter - 1
Более наглядная форма:

counter = counter + 1

6.8 Типичные ошибки

1. Деление на ноль.
2. Использование целочисленного типа при необходимости дробного результата.
3. Отсутствие преобразования числа в строку при выводе.

Практическое задание

1. Создайте программу, которая:
    - создаёт две переменные;
    - выполняет все арифметические операции;
    - выводит результат в одном окне.
2. Измените тип переменных на .d и сравните результат деления.

Итоги главы

В данной главе были рассмотрены:

- основные типы данных;
- арифметические операции;
- порядок вычислений;
- особенности деления;
- преобразование чисел в строки.

В следующей главе будет рассмотрена тема логики в программе — условные конструкции.

Глава 7

Условные конструкции (If)

7.1 Назначение условных конструкций

В реальных программах необходимо принимать решения.

Пример:

- Если пользователь ввёл правильный пароль — открыть доступ.
- Если число больше нуля — выполнить вычисление.
- Если файл существует — открыть его.

Для реализации таких проверок используются условные конструкции.

В языке **PureBasic** применяется оператор If.

7.2 Простая форма оператора If

Общий синтаксис:

```
If условие
  ; команды
EndIf
```

Если условие истинно (True), выполняются команды внутри блока.
Если условие ложно (False), блок пропускается.

7.3 Простейший пример

```
age.i = 18

If age >= 18
  MessageRequester("Проверка", "Доступ разрешён")
EndIf
```

Если переменная age больше или равна 18, будет показано сообщение.

7.4 Операторы сравнения

Оператор	Значение
=	равно
<>	не равно
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

Пример:

```
number.i = 10

If number <> 5
  MessageRequester("Проверка", "Число не равно 5")
EndIf
```

7.5 Конструкция If…Else

Если необходимо выполнить альтернативное действие:

```
If условие
  ; выполняется, если условие истинно
Else
  ; выполняется, если условие ложно
EndIf
```

Пример:

```
temperature.i = 5

If temperature > 0
  MessageRequester("Погода", "Температура положительная")
Else
  MessageRequester("Погода", "Температура нулевая или отрицательная")
EndIf
```

7.6 Конструкция If…ElseIf…Else

Если необходимо проверить несколько условий:

```
score.i = 75

If score >= 90
  MessageRequester("Оценка", "Отлично")
ElseIf score >= 70
  MessageRequester("Оценка", "Хорошо")
ElseIf score >= 50
  MessageRequester("Оценка", "Удовлетворительно")
Else
  MessageRequester("Оценка", "Неудовлетворительно")
EndIf
```

Условия проверяются сверху вниз.
Как только найдено истинное условие, остальные не проверяются.

7.7 Логические операторы

Иногда требуется проверить несколько условий одновременно.

Оператор	Назначение
`And`	логическое «И»
`Or`	логическое «ИЛИ»
`Not`	логическое отрицание

Пример:

```
 age.i = 20
hasTicket.i = 1

If age >= 18 And hasTicket = 1
  MessageRequester("Вход", "Проход разрешён")
EndIf
```

7.8 Вложенные условия

Условие может находиться внутри другого условия:

```
 age.i = 25

If age >= 18
  If age >= 60
    MessageRequester("Категория", "Пенсионер")
  Else
    MessageRequester("Категория", "Взрослый")
  EndIf
EndIf
```

7.9 Типичные ошибки

1. Пропущенный EndIf.
2. Использование одного знака = вместо оператора сравнения (в PureBasic это допустимо, но важно понимать контекст).
3. Логические ошибки в порядке условий.
4. Отсутствие фигурной структуры (неаккуратное форматирование).

Практическое задание

1. Создайте программу, которая:
    - проверяет возраст;
    - выводит разные сообщения для:
        - младше 18;
        - от 18 до 60;
        - старше 60.
2. Добавьте дополнительную проверку с использованием And.

Итоги главы

В данной главе были рассмотрены:

- оператор If;
- конструкции Else и ElseIf;
- операторы сравнения;
- логические операторы;
- вложенные условия.

В следующей главе будет рассмотрена тема повторяющихся действий — циклы.

Глава 8

Циклы

8.1 Назначение циклов

В программировании часто требуется выполнять одно и то же действие несколько раз.

Примеры:

- вывести числа от 1 до 10;
- обработать список данных;
- повторять действия до выполнения условия.

Для этого используются циклы.

В языке **PureBasic** существуют несколько видов циклов.

8.2 Цикл For…Next

Цикл For применяется, когда известно количество повторений.

Синтаксис:

```
For переменная = начальное_значение To конечное_значение
  ; команды
Next
```

Пример:

```
For i = 1 To 5
  MessageRequester("Цикл For", "Значение i = " + Str(i))
Next
```

Цикл выполнится 5 раз.

Изменение шага цикла

По умолчанию шаг равен 1.
Можно указать другой шаг с помощью Step:

```
For i = 1 To 10 Step 2
Debug i
 Next
```

8.3 Цикл While…Wend

Используется, когда заранее неизвестно количество повторений.

Синтаксис:

While условие
  ; команды
Wend
Цикл выполняется, пока условие истинно.

Пример:

```
counter.i = 1

While counter <= 5
  Debug counter
  counter = counter + 1
Wend
```

Важно изменять переменную внутри цикла, иначе возникнет бесконечный цикл.

8.4 Цикл Repeat…Until

Этот цикл выполняется минимум один раз.

Синтаксис:

Repeat
  ; команды
Until условие
Цикл повторяется до тех пор, пока условие не станет истинным.

Пример:

```
number.i = 1

Repeat
  Debug number
  number = number + 1
Until number > 5
```

8.5 Операторы управления циклом

Break — выход из цикла

```
For i = 1 To 10
  If i = 5
    Break
  EndIf
  Debug i
Next
```

Цикл завершится при значении i = 5.

Continue — переход к следующей итерации

```
For i = 1 To 5
If i = 3
Continue
EndIf
Debug i
Next
```

Число 3 не будет выведено.

8.6 Сравнение циклов

Цикл	Когда использовать
For	известно число повторений
While	повторение до выполнения условия
Repeat	действие должно выполниться минимум один раз

8.7 Типичные ошибки

1. Бесконечный цикл (условие никогда не становится ложным).
2. Отсутствие изменения счётчика.
3. Неверные границы цикла.

Пример бесконечного цикла:

```
While 1 = 1
Debug "Бесконечно"
Wend
```

Практическое задание

1. Напишите программу, которая выводит числа от 1 до 20.
2. Выведите только чётные числа.
3. Создайте цикл, который завершится при достижении значения 10 с использованием Break.

Итоги главы

В данной главе были рассмотрены:

- цикл For;
- цикл While;
- цикл Repeat;
- операторы Break и Continue;
- типичные ошибки при работе с циклами.

В следующей главе будет рассмотрено важное понятие структурирования программы — процедуры.

Глава 9

Процедуры

9.1 Назначение процедур

По мере увеличения объёма программы код становится более сложным.
Для упрощения структуры используются процедуры.

Процедура — это именованный блок кода, который выполняет определённую задачу и может быть вызван из любого места программы.

В языке **PureBasic** процедуры позволяют:

- избегать повторения кода;
- улучшать читаемость программы;
- структурировать проект;
- повторно использовать логические блоки.

9.2 Объявление процедуры

Общий синтаксис

```
Procedure ИмяПроцедуры()
  ; команды
EndProcedure
```

Простейший пример

```
Procedure ShowMessage()
  MessageRequester("Сообщение", "Это процедура")
EndProcedure

ShowMessage()
```

Разбор

1. Объявляется процедура ShowMessage.
2. После объявления она вызывается по имени.
3. Код внутри процедуры выполняется при вызове.

9.3 Процедуры с параметрами

Процедура может принимать данные.

Пример

```
Procedure ShowName(name.s)
  MessageRequester("Имя", name)
EndProcedure

ShowName("Алексей")
ShowName("Мария")
```

Разбор

- name.s — параметр типа строка.
- При каждом вызове передаётся новое значение.

9.4 Возврат значения из процедуры

Процедура может возвращать результат с помощью ProcedureReturn.

Пример

```
Procedure.i AddNumbers(a.i, b.i)
  result = a + b
  ProcedureReturn result
EndProcedure

sum = AddNumbers(5, 7)
MessageRequester("Результат", Str(sum))
```

Объяснение

- .i после имени процедуры означает, что она возвращает целое число.
- ProcedureReturn завершает выполнение и возвращает значение.

9.5 Локальные и глобальные переменные

Локальные переменные

Переменные, объявленные внутри процедуры, доступны только внутри неё.

```
Procedure Test()
  value.i = 10
EndProcedure
```

Глобальные переменные

Объявляются с помощью Global и доступны во всей программе.

```
Global counter.i
```

Использование глобальных переменных следует минимизировать, чтобы избежать логических ошибок.

9.6 Порядок размещения процедур

Процедуры рекомендуется размещать:

- либо в начале файла;
- либо в отдельной логической области программы.

Это улучшает читаемость кода.

9.7 Типичные ошибки

1. Отсутствие EndProcedure.
2. Несоответствие типа возвращаемого значения.
3. Попытка использовать локальную переменную вне процедуры.
4. Ошибка в количестве переданных параметров.

Практическое задание

1. Создайте процедуру, которая:
    - принимает два числа;
    - возвращает их произведение.
2. Создайте процедуру, которая:
    - принимает имя;
    - выводит персональное приветствие.

Итоги главы

В данной главе были рассмотрены:

- назначение процедур;
- объявление и вызов;
- параметры;
- возврат значения;
- локальные и глобальные переменные.

На данном этапе завершена базовая часть языка.

В следующей главе начнётся работа со строками и текстовыми данными.

Глава 10

Строки и работа с текстом

10.1 Понятие строки

Строка — это последовательность символов, заключённых в двойные кавычки.

Примеры строк:

```
"Здравствуйте"
"PureBasic"
"2026"
```

В языке **PureBasic** строковый тип обозначается суффиксом .s.

10.2 Объявление строковой переменной

```
name.s = "Алексей"
city.s = "Москва"
```

Строка может содержать:

- буквы;
- цифры;
- пробелы;
- специальные символы.

10.3 Объединение строк (конкатенация)

Для объединения строк используется оператор +.

```
firstName.s = "Иван"
lastName.s = "Петров"

fullName.s = firstName + " " + lastName

MessageRequester("Пользователь", fullName)
```

10.4 Добавление числа к строке

Число необходимо предварительно преобразовать в строку с помощью функции Str().

```
age.i = 25
text.s = "Возраст: " + Str(age)

MessageRequester("Информация", text)
```

10.5 Длина строки

Для определения количества символов используется функция Len().

```
text.s = "PureBasic"
length.i = Len(text)

MessageRequester("Длина", Str(length))
```

10.6 Извлечение части строки

Функция Left()

Возвращает символы с начала строки.

```
text.s = "Программирование"
result.s = Left(text, 6)
```

Функция Right()

Возвращает символы с конца строки.

```
result.s = Right(text, 6)
```

Функция Mid()

Возвращает часть строки с указанной позиции.

```
result.s = Mid(text, 3, 5)
```

Параметры:

1. Строка
2. Начальная позиция
3. Количество символов

10.7 Поиск в строке

Функция FindString() позволяет найти подстроку.

```
text.s = "Я изучаю PureBasic"

position.i = FindString(text, "PureBasic")

If position > 0
  MessageRequester("Результат", "Слово найдено")
EndIf
```

Если строка не найдена, функция возвращает 0.

10.8 Изменение регистра

```
text.s = "PureBasic"

upper.s = UCase(text)  ; ВЕРХНИЙ РЕГИСТР
lower.s = LCase(text)  ; нижний регистр
```

10.9 Удаление пробелов

```
text.s = "   пример текста   "

cleanText.s = Trim(text)
```

10.10 Замена части строки

Функция ReplaceString() позволяет заменить текст.

```
text.s = "Я изучаю Basic"
newText.s = ReplaceString(text, "Basic", "PureBasic")
```

10.11 Перевод строки

Для добавления переноса строки используется константа:

#CRLF$
Пример:

```
text.s = "Строка 1" + #CRLF$ + "Строка 2"

MessageRequester("Текст", text)
```

10.12 Типичные ошибки

1. Попытка объединить число и строку без Str().
2. Неверная позиция в Mid().
3. Ошибка при поиске строки (не учтён регистр символов).

Практическое задание

1. Создайте программу, которая:
    - запрашивает имя пользователя;
    - выводит длину имени;
    - выводит имя в верхнем регистре.
2. Создайте строку и замените одно слово другим.

Итоги главы

В данной главе были рассмотрены:

- создание строк;
- объединение текста;
- преобразование чисел;
- функции работы со строками;

- поиск и замена текста.

В следующей главе будет рассмотрена работа с массивами — хранение нескольких значений в одной структуре.

Глава 11

Массивы

11.1 Назначение массивов

В предыдущих главах использовались переменные для хранения одного значения. Однако на практике часто требуется хранить **несколько однотипных значений**.

Примеры:

- список оценок;
- набор чисел;
- перечень имён;
- результаты измерений.

Для хранения таких данных используются массивы.

В языке **PureBasic** массив позволяет объединить несколько значений под одним именем.

11.2 Объявление массива

Синтаксис

Dim имя(размер)
Пример:

Dim numbers(4)
Важно:
Если указано число 4, фактически создаются элементы с индексами от 0 до 4 (всего 5 элементов).

Индексация в PureBasic начинается с 0.

11.3 Заполнение массива

```
Dim numbers(4)

numbers(0) = 10
numbers(1) = 20
numbers(2) = 30
numbers(3) = 40
numbers(4) = 50
```

11.4 Доступ к элементам массива

Чтобы получить значение элемента, указывается его индекс:

```
MessageRequester("Значение", Str(numbers(2)))
```

В данном случае будет выведено число 30.

11.5 Использование цикла с массивом

Массивы обычно обрабатываются с помощью цикла.

```
Dim numbers(4)
For i = 0 To 4
numbers(i) = i * 10
Next
For i = 0 To 4
Debug numbers(i)
Next
```

11.6 Массивы строк

Можно создавать массивы строк:

```
Dim names.s(2)

names(0) = "Анна"
names(1) = "Иван"
```

names(2) = "Мария"

11.7 Многомерные массивы

PureBasic поддерживает многомерные массивы.

Пример двумерного массива:

```
Dim matrix(2, 2)

matrix(0,0) = 1
matrix(0,1) = 2
matrix(1,0) = 3
matrix(1,1) = 4
```

11.8 Определение размера массива

Функция ArraySize() позволяет определить максимальный индекс массива.

```
size = ArraySize(numbers())
```
Если массив объявлен как Dim numbers(4), функция вернёт 4.

11.9 Изменение размера массива

Размер можно изменить с помощью ReDim.

```
ReDim numbers(10)
```
Следует учитывать, что при изменении размера данные могут быть перезаписаны.

11.10 Типичные ошибки

1. Обращение к несуществующему индексу.
2. Забытый вызов Dim.
3. Неправильное понимание того, что индексация начинается с 0.
4. Потеря данных при использовании ReDim.

Практическое задание

1. Создайте массив из 10 чисел.

2. Заполните его значениями от 1 до 10.
3. Выведите сумму всех элементов массива.
4. Создайте двумерный массив 3×3 и заполните его числами.

Итоги главы

В данной главе были рассмотрены:

- назначение массивов;
- объявление и заполнение;
- использование циклов;
- строковые массивы;
- многомерные массивы;
- изменение размера массива.

В следующей главе будет рассмотрена более гибкая структура хранения данных — списки.

Глава 12

Списки

12.1 Назначение списков

Массивы удобны, когда заранее известно количество элементов.
Однако в реальных программах размер набора данных часто изменяется во время работы.

Для таких случаев используются списки.

В языке **PureBasic** список — это динамическая структура данных, размер которой автоматически изменяется.

12.2 Объявление списка

Синтаксис

NewList имя.тип()
Пример:

NewList numbers.i()
Создан пустой список целых чисел.

12.3 Добавление элементов

Для добавления элемента используется команда AddElement().

AddElement(numbers())
numbers() = 10

AddElement(numbers())
numbers() = 20

AddElement(numbers())
numbers() = 30

Каждый вызов AddElement() создаёт новый элемент.

12.4 Перебор списка

Для просмотра всех элементов используется цикл ForEach.

ForEach numbers()
  Debug numbers()
Next
Цикл автоматически проходит по всем элементам списка.

12.5 Удаление элементов

Удаление текущего элемента:

DeleteElement(numbers())
Удаление всех элементов:

ClearList(numbers())

12.6 Подсчёт элементов

Функция ListSize() возвращает количество элементов:

```
count = ListSize(numbers())
```

12.7 Списки строк

```
NewList names.s()

AddElement(names())
names() = "Анна"

AddElement(names())
names() = "Иван"

ForEach names()
  Debug names()
Next
```

12.8 Переход к конкретному элементу

Можно перемещаться по списку вручную:

```
FirstElement(numbers())
NextElement(numbers())
PreviousElement(numbers())
LastElement(numbers())
```

12.9 Сравнение массивов и списков

Характеристика	Массив	Список
Размер фиксирован	Да	Нет
Изменение размера	Через ReDim	Автоматически
Удобство добавления	Среднее	Высокое
Скорость доступа по индексу	Высокая	Ниже

12.10 Типичные ошибки

1. Попытка обратиться к элементу без AddElement().

2. Использование списка без предварительного объявления NewList.
3. Ошибки при удалении элемента во время перебора.

Практическое задание

1. Создайте список из 5 чисел.
2. Выведите их сумму.
3. Удалите один элемент и снова выведите список.
4. Создайте список строк и выведите их в алфавитном порядке (подсказка: используйте SortList()).

Итоги главы

В данной главе были рассмотрены:

- назначение списков;
- добавление и удаление элементов;
- перебор с помощью ForEach;
- различия между массивами и списками.

В следующей главе будет рассмотрена структура данных, объединяющая несколько полей — структуры.

Глава 13

Структуры

13.1 Назначение структур

В предыдущих главах рассматривались переменные, массивы и списки.
Однако часто требуется хранить **несколько связанных значений вместе**.

Пример:

Информация о человеке:

- имя

- возраст
- город

Создавать отдельные переменные неудобно.
Для объединения связанных данных используются структуры.

В языке **PureBasic** структура позволяет создать собственный тип данных.

13.2 Объявление структуры

Синтаксис

```
Structure ИмяСтруктуры
  поле1.тип
  поле2.тип
  поле3.тип
EndStructure
```

Пример

```
Structure Person
name.s
age.i
city.s
EndStructure
```

Создан новый тип данных Person.

13.3 Создание переменной структуры

```
Define user.Person
```
Теперь переменная user содержит три поля:

- user\name
- user\age
- user\city

13.4 Заполнение структуры

```
user\name = "Анна"
user\age = 28
user\city = "Москва"
```

13.5 Использование структуры

```
MessageRequester("Информация",
  "Имя: " + user\name + #CRLF$ +
  "Возраст: " + Str(user\age) + #CRLF$ +
  "Город: " + user\city)
```

13.6 Структуры в списках

Структуры особенно удобны в списках

```
NewList people.Person()

AddElement(people())
people()\name = "Иван"
people()\age = 30
people()\city = "Казань"

AddElement(people())
people()\name = "Мария"
people()\age = 25
people()\city = "Сочи"
Перебор:


ForEach people()
  Debug people()\name + " - " + Str(people()\age)
Next
```

13.7 Структуры в массивах

```
Dim employees.Person(2)
```

employees(0)\name = "Алексей"
employees(0)\age = 35

13.8 Вложенные структуры

Можно создавать структуры внутри структур.

```
Structure Address
  city.s
  street.s
EndStructure

Structure Client
  name.s
  address.Address
EndStructure
```

Доступ к вложенному полю:

client\address\city = "Москва"

13.9 Практическое применение

Структуры используются для:

- хранения записей (клиенты, товары, сотрудники);
- работы с базами данных;
- передачи нескольких параметров в процедуры;
- системного программирования.

13.10 Типичные ошибки

1. Обращение к несуществующему полю.
2. Отсутствие EndStructure.
3. Неправильное использование слэша \ при доступе к полям.
4. Забытое преобразование числовых полей в строку при выводе.

Практическое задание

1. Создайте структуру Book:
    - название;
    - автор;
    - год издания.
2. Создайте список книг.
3. Выведите все книги.
4. Добавьте вложенную структуру Publisher.

Итоги главы

В данной главе были рассмотрены:

- создание структур;
- работа с полями;
- использование в списках и массивах;
- вложенные структуры.

В следующей главе будет рассмотрена структура данных для хранения пар «ключ–значение» — Map (словарь).

Глава 14

Map (Словари)

14.1 Назначение структуры Map

В предыдущих главах рассматривались:

- массивы — доступ по индексу;
- списки — динамическое хранение;
- структуры — объединение полей.

Однако иногда требуется получать данные **по имени (ключу)**, а не по числовому индексу.

Пример:

- по логину найти пользователя;
- по коду товара найти цену;
- по слову получить перевод.

Для таких задач используется структура **Map** (словарь).

В языке **PureBasic** Map хранит данные в формате:

ключ → значение

14.2 Объявление Map

Синтаксис

NewMap имя.тип()
Пример:

NewMap ages.i()
Создан словарь, где:

- ключ — строка;
- значение — целое число.

Важно:
В PureBasic ключом всегда является строка.

14.3 Добавление элементов

Добавление элемента происходит через указание ключа в скобках

```
ages("Иван") = 30
ages("Мария") = 25
ages("Анна") = 28
```

Если ключа не существует, он создаётся автоматически.

14.4 Доступ к элементу

```
MessageRequester("Возраст", Str(ages("Иван")))
```

14.5 Проверка существования ключа

Перед обращением рекомендуется проверять наличие ключа:

```
If FindMapElement(ages(), "Мария")
  MessageRequester("Результат", "Возраст: " + Str(ages()))
Else
  MessageRequester("Ошибка", "Ключ не найден")
EndIf
```

14.6 Перебор Map

Для перебора используется ForEach:

```
ForEach ages()
  Debug MapKey(ages()) + " - " + Str(ages())
Next
```

Функция MapKey() возвращает текущий ключ.

14.7 Удаление элементов

Удаление одного элемента:

```
DeleteMapElement(ages())
```

Удаление всех элементов:

```
ClearMap(ages())
```

14.8 Map со структурами

Map может хранить сложные типы данных.

```
Structure Person
  name.s
  age.i
EndStructure

NewMap people.Person()

people("user1")\name = "Алексей"
people("user1")\age = 35
```

14.9 Сравнение Map с другими структурами

Структура	Доступ	Размер	Применение
Массив	По индексу	Фиксированный	Списки чисел
Список	Последовательный	Динамический	Перебор данных
Map	По ключу	Динамический	Поиск по имени

14.10 Типичные ошибки

1. Попытка использовать числовой ключ.
2. Отсутствие проверки FindMapElement().
3. Удаление элемента вне текущей позиции.
4. Неверное понимание, что ключ всегда строковый.

Практическое задание

1. Создайте Map для хранения оценок студентов.
2. Добавьте минимум 5 записей.
3. Проверьте наличие одного из студентов.
4. Выведите все элементы словаря.
5. Удалите одну запись.

Итоги главы

В данной главе были рассмотрены:

- назначение Map;
- добавление и поиск элементов;
- перебор ключей;
- удаление элементов;
- использование структур в Map.

На этом завершается раздел по структурам хранения данных.

В следующей главе начнётся работа с файлами — сохранение и чтение данных.

Глава 15

Работа с файлами

15.1 Назначение работы с файлами

До настоящего момента все данные существовали только во время работы программы. После её завершения информация теряется.

Для сохранения данных используются файлы.

Работа с файлами позволяет:

- сохранять результаты вычислений;
- хранить настройки программы;
- вести журналы событий;
- загружать данные при запуске.

В языке **PureBasic** предусмотрены встроенные средства для чтения и записи файлов.

15.2 Открытие файла для записи

Синтаксис

```
If CreateFile(номер, "имя_файла.txt") ; запись
  CloseFile(номер)
EndIf
```

Пример:

```
If CreateFile(0, "test.txt")
  WriteStringN(0, "Первая строка")
  WriteStringN(0, "Вторая строка")
  CloseFile(0)
EndIf
```

Объяснение

- CreateFile() создаёт новый файл.
- 0 — номер файла (идентификатор).
- WriteStringN() записывает строку с переводом строки.
- CloseFile() закрывает файл.

15.3 Открытие файла для чтения

```
If ReadFile(0, "test.txt")

  While Eof(0) = 0
    line.s = ReadString(0)
    Debug line
  Wend
  CloseFile(0)
EndIf
```

Разбор

- ReadFile() открывает файл.
- Eof() проверяет конец файла.
- ReadString() читает строку.

15.4 Добавление данных в существующий файл

```
If OpenFile(0, "test.txt")
  FileSeek(0, Lof(0))
  WriteStringN(0, "Новая строка")
  CloseFile(0)
EndIf
```

- OpenFile() открывает существующий файл.
- FileSeek() перемещает указатель в конец.
- Lof() возвращает размер файла.

15.5 Проверка существования файла

```
If FileSize("test.txt") > 0
  MessageRequester("Проверка", "Файл существует")
Else
  MessageRequester("Проверка", "Файл не найден")
EndIf
```

15.6 Работа с числами

Можно записывать числа:

```
If CreateFile(0, "numbers.txt")
  WriteStringN(0, Str(100))
  CloseFile(0)
EndIf
```

При чтении необходимо преобразовать строку обратно в число:

value.i = Val(ReadString(0))

15.7 Пример полной программы

```
; Запись
If CreateFile(0, "users.txt")
  WriteStringN(0, "Иван")
  WriteStringN(0, "Мария")
  CloseFile(0)
EndIf

; Чтение
If ReadFile(0, "users.txt")
  While Eof(0) = 0
    name.s = ReadString(0)
    MessageRequester("Пользователь", name)
  Wend
  CloseFile(0)
EndIf
```

15.8 Типичные ошибки

1. Забытый CloseFile().
2. Попытка читать несуществующий файл.
3. Неправильный номер файла.
4. Отсутствие проверки If.

Практическое задание

1. Создайте программу, которая:
    - записывает 5 имён в файл.
2. Затем:
    - считывает файл;
    - выводит имена на экран.
3. Добавьте в файл ещё одну запись без удаления старых данных.

Итоги главы

В данной главе были рассмотрены:

- создание файла;
- запись строк;
- чтение данных;
- добавление информации;
- проверка существования файла.

На этом завершается раздел базовых возможностей языка.

В следующей части книги начнётся работа с графическим интерфейсом — создание оконных приложений.

Глава 16

Создание окна

16.1 Понятие графического интерфейса

Графический интерфейс пользователя (GUI) позволяет взаимодействовать с программой с помощью:

- окон;
- кнопок;
- полей ввода;
- меню;
- списков.

В языке **PureBasic** создание оконных приложений осуществляется с использованием встроенных средств.

Программы используют нативные элементы управления операционной системы, благодаря чему интерфейс выглядит естественно.

16.2 Простейшее окно

Пример

```
OpenWindow(0, 100, 100, 400, 300, "Моё первое окно")

Repeat
  event = WaitWindowEvent()
Until event = #PB_Event_CloseWindow
```

16.3 Разбор программы

1. OpenWindow()

```
OpenWindow(номер, x, y, ширина, высота, заголовок)
```
Параметры:

- номер окна (идентификатор);
- координаты положения на экране;
- размеры;
- текст заголовка.

2. Цикл обработки событий

```
Repeat
  event = WaitWindowEvent()
Until event = #PB_Event_CloseWindow
```
Окно работает до тех пор, пока пользователь его не закроет.

Если не добавить этот цикл, окно закроется сразу после запуска.

16.4 Центрирование окна

Можно использовать специальный флаг:

```
OpenWindow(0, #PB_Ignore, #PB_Ignore, 400, 300, "Окно",
          #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
```
Флаг #PB_Window_ScreenCentered размещает окно по центру экрана.

16.5 Добавление минимальной структуры проекта

Рекомендуемый шаблон:

```
If OpenWindow(0, #PB_Ignore, #PB_Ignore, 400, 300,
              "Программа",
              #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

  Repeat
    event = WaitWindowEvent()
  Until event = #PB_Event_CloseWindow

EndIf
```

Проверка If гарантирует, что окно успешно создано.

16.6 Несколько окон

Можно создавать несколько окон, указывая разные номера:

```
OpenWindow(0, 100, 100, 300, 200, "Окно 1")
OpenWindow(1, 200, 200, 300, 200, "Окно 2")
```

Каждое окно имеет свой идентификатор.

16.7 Типичные ошибки

1. Отсутствие цикла обработки событий.
2. Использование одинакового номера для разных окон.
3. Неправильные флаги окна.
4. Отсутствие проверки успешного создания.

Практическое задание

1. Создайте окно размером 500×400.
2. Разместите его по центру экрана.
3. Измените заголовок.
4. Попробуйте создать два окна одновременно.

Итоги главы

В данной главе были рассмотрены:

- создание окна;
- параметры функции OpenWindow();
- обработка событий;
- центрирование окна;
- создание нескольких окон.

В следующей главе будет рассмотрено добавление элементов управления — кнопок, полей ввода и других компонентов интерфейса.

Глава 17

Элементы управления (Gadgets)

17.1 Понятие элементов управления

Окно само по себе не обеспечивает взаимодействия с пользователем.
Для ввода данных и управления программой используются элементы управления — **gadgets**.

В языке **PureBasic** элементы управления создаются с помощью специальных функций.

К наиболее распространённым относятся:

- кнопки;
- текстовые надписи;
- поля ввода;
- флажки;
- списки.

17.2 Кнопка (ButtonGadget)

Пример

```
If OpenWindow(0, #PB_Ignore, #PB_Ignore, 400, 200,
          "Пример",
          #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

  ButtonGadget(0, 150, 80, 100, 30, "Нажми меня")

  Repeat
    event = WaitWindowEvent()
  Until event = #PB_Event_CloseWindow
```

Параметры

ButtonGadget(номер, x, y, ширина, высота, текст)

17.3 Надпись (TextGadget)

Используется для отображения текста.

```
TextGadget(1, 20, 20, 200, 25, "Введите имя:")
```

17.4 Поле ввода (StringGadget)

Позволяет пользователю вводить текст.

```
StringGadget(2, 20, 50, 200, 25, "")
```

Получение текста:

```
name.s = GetGadgetText(2)
```

17.5 Обработка нажатия кнопки

Для обработки действий используется цикл событий.

```
If OpenWindow(0, #PB_Ignore, #PB_Ignore, 400, 200,
          "Пример",
          #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

 TextGadget(1, 20, 20, 200, 25, "Введите имя:")
 StringGadget(2, 20, 50, 200, 25, "")
 ButtonGadget(3, 20, 90, 100, 30, "Приветствие")

 Repeat
  event = WaitWindowEvent()

  If event = #PB_Event_Gadget
   If EventGadget() = 3
    name.s = GetGadgetText(2)
    MessageRequester("Здравствуйте", "Привет, " + name + "!")
   EndIf
  EndIf

 Until event = #PB_Event_CloseWindow

EndIf
```

17.6 Флажок (CheckBoxGadget)

```
CheckBoxGadget(4, 20, 130, 200, 25, "Согласен с условиями")
```

Проверка состояния:

```
 If GetGadgetState(4) = 1
 MessageRequester("Статус", "Флажок установлен")
EndIf
```

17.7 Список (ListViewGadget)

```
ListViewGadget(5, 250, 20, 120, 120)
AddGadgetItem(5, -1, "Пункт 1")
AddGadgetItem(5, -1, "Пункт 2")
Получение выбранного элемента:


selected = GetGadgetState(5)
```

17.8 Организация идентификаторов

Рекомендуется объявлять идентификаторы в начале программы:

```
Enumeration
  #Window_Main
  #Text_Label
  #String_Input
  #Button_Ok
EndEnumeration
```

Это повышает читаемость кода.

17.9 Типичные ошибки

1. Использование одинакового номера для разных элементов.
2. Отсутствие проверки #PB_Event_Gadget.
3. Попытка получить текст до создания элемента.
4. Неверный идентификатор в GetGadgetText().

Практическое задание

1. Создайте окно с:
    - надписью;
    - полем ввода;
    - кнопкой.
2. При нажатии кнопки:
    - вывести введённый текст.

3. Добавьте флажок и измените поведение программы в зависимости от его состояния.

Итоги главы

В данной главе были рассмотрены:

- создание кнопок и полей ввода;
- получение данных от пользователя;
- обработка событий;
- использование флажков и списков;
- организация идентификаторов.

В следующей главе будет подробно рассмотрена система событий и управление интерфейсом.

Глава 18

Обработка событий

18.1 Понятие событийной модели

Графическое приложение работает по событийной модели.

Это означает, что программа:

1. Ожидает действия пользователя.
2. Получает событие.
3. Реагирует на него соответствующим образом.

События могут быть вызваны:

- нажатием кнопки;
- закрытием окна;
- изменением текста;
- выбором элемента списка;

- перемещением мыши.

В языке **PureBasic** обработка событий осуществляется через цикл ожидания.

18.2 Основной цикл обработки событий

Базовая структура:

```
Repeat
  event = WaitWindowEvent()
  Select event
    Case #PB_Event_CloseWindow
      Break
  EndSelect
ForEver
```

WaitWindowEvent() ожидает событие и возвращает его код.

18.3 Обработка событий элементов управления

Для обработки действий, связанных с элементами управления, используется событие:

#PB_Event_Gadget
Пример:

```
If OpenWindow(0, #PB_Ignore, #PB_Ignore, 400, 200,
              "События",
              #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

  ButtonGadget(0, 150, 80, 100, 30, "Нажать")
  Repeat
    event = WaitWindowEvent()

    Select event
     Case #PB_Event_Gadget

       Select EventGadget()
         Case 0
          MessageRequester("Событие", "Кнопка нажата")
       EndSelect

     Case #PB_Event_CloseWindow
       Break
    EndSelect

  ForEver
EndIf
```

18.4 Функции для работы с событиями

EventGadget()

Возвращает номер элемента управления, вызвавшего событие.

EventType()

Позволяет определить тип события элемента управления.

Например:

```
If EventType() = #PB_EventType_LeftClick
```

18.5 Обработка изменения текста

Пример реакции на изменение содержимого поля ввода:

```
StringGadget(1, 20, 50, 200, 25, "")

Repeat
  event = WaitWindowEvent()
  If event = #PB_Event_Gadget And EventGadget() = 1
    If EventType() = #PB_EventType_Change
      Debug "Текст изменён"
    EndIf
  EndIf

Until event = #PB_Event_CloseWindow
```

18.6 Использование Select вместо вложенных If

Для повышения читаемости рекомендуется использовать Select:

```
Select EventGadget()
  Case #Button_Ok
    ; действия
  Case #Button_Cancel
    ; действия
EndSelect
```

18.7 Обработка нескольких окон

Если программа содержит несколько окон, можно использовать:

EventWindow()
Она возвращает номер окна, в котором произошло событие.

18.8 Рекомендуемая структура программы

```
Enumeration
  #Window_Main
  #Button_Ok
EndEnumeration

If OpenWindow(#Window_Main, #PB_Ignore, #PB_Ignore, 400, 200,
              "Пример",
              #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

  ButtonGadget(#Button_Ok, 150, 80, 100, 30, "OK")
  Repeat
    event = WaitWindowEvent()
    Select event

      Case #PB_Event_Gadget
        Select EventGadget()
          Case #Button_Ok
            MessageRequester("Событие", "Нажата кнопка OK")
        EndSelect

      Case #PB_Event_CloseWindow
        Break

    EndSelect
  ForEver
EndIf
```

18.9 Типичные ошибки

1. Отсутствие цикла обработки событий.

2. Игнорирование EventType().
3. Использование числовых идентификаторов вместо Enumeration.
4. Отсутствие обработки закрытия окна.

Практическое задание

1. Создайте окно с двумя кнопками.
2. Каждая кнопка должна выводить разное сообщение.
3. Добавьте поле ввода и реагируйте на изменение текста.
4. Используйте Enumeration для всех элементов.

Итоги главы

В данной главе были рассмотрены:

- событийная модель;
- основной цикл обработки событий;
- обработка нажатий;
- использование EventGadget() и EventType();
- структурирование кода.

В следующей главе будет рассмотрено создание меню и диалоговых окон.

Глава 19

Меню и диалоговые окна

19.1 Назначение меню

Меню — это элемент интерфейса, предоставляющий пользователю доступ к командам программы.

Стандартное приложение обычно содержит:

- главное меню (в верхней части окна);

- контекстное меню (по правому клику);
- системные диалоговые окна.

В языке **PureBasic** создание меню осуществляется с помощью встроенных функций.

19.2 Создание главного меню

Примеры:

```
Enumeration
  #Window_Main
  #Menu_File
  #Menu_Exit
EndEnumeration

If OpenWindow(#Window_Main, #PB_Ignore, #PB_Ignore, 500, 300,
              "Меню",
              #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

  If CreateMenu(0, WindowID(#Window_Main))

    MenuTitle("Файл")
    MenuItem(#Menu_Exit, "Выход")
  EndIf

  Repeat
    event = WaitWindowEvent()

    Select event
      Case #PB_Event_Menu
        Select EventMenu()
          Case #Menu_Exit
            Break
        EndSelect

      Case #PB_Event_CloseWindow
        Break

    EndSelect

  ForEver

EndIf
```

19.3 Добавление нескольких пунктов меню

```
MenuTitle("Файл")
MenuItem(#Menu_Open, "Открыть")
MenuItem(#Menu_Save, "Сохранить")
MenuBar()
MenuItem(#Menu_Exit, "Выход")
```

- MenuBar() добавляет разделительную линию.

19.4 Контекстное меню

Контекстное меню вызывается при правом клике мыши.

```
If CreatePopupMenu(1)
  MenuItem(1, "Копировать")
  MenuItem(2, "Вставить")
EndIf
```

Отображение:

```
DisplayPopupMenu(1, WindowID(#Window_Main))
```

Обычно вызывается при событии правого клика.

19.5 Диалоговые окна

PureBasic предоставляет встроенные диалоги.

Открытие файла

```
file.s = OpenFileRequester("Выберите файл", "",
                           "Текстовые файлы (*.txt)|*.txt", 0)
```

Сохранение файла

```
file.s = SaveFileRequester("Сохранить файл", "",
                           "Текстовые файлы (*.txt)|*.txt", 0)
```

Выбор папки

folder.s = PathRequester("Выберите папку", "")

Выбор цвета

color = ColorRequester()

19.6 Обработка событий меню

Для меню используется событие:

#PB_Event_Menu
Получение выбранного пункта:

EventMenu()

19.7 Рекомендуемая организация идентификаторов

```
Enumeration
  #Window_Main

  #Menu_File
  #Menu_Open
  #Menu_Save
  #Menu_Exit
EndEnumeration
```

Это упрощает сопровождение проекта.

19.8 Типичные ошибки

1. Отсутствие CreateMenu().
2. Необработанное событие #PB_Event_Menu.
3. Повторяющиеся идентификаторы.
4. Попытка вызвать диалог вне цикла событий.

Практическое задание

1. Создайте программу с меню:

    - Файл → Открыть;
    - Файл → Сохранить;
    - Файл → Выход.
2. Реализуйте вызов диалога выбора файла.
3. Добавьте контекстное меню.
4. Используйте Enumeration.

Итоги главы

В данной главе были рассмотрены:

- создание главного меню;
- добавление пунктов;
- контекстное меню;
- встроенные диалоговые окна;
- обработка событий меню.

На этом завершается базовый раздел по созданию интерфейса.

В следующей главе начнётся первый практический проект — создание калькулятора.

Глава 20

Мини-проект: Калькулятор

20.1 Постановка задачи

Цель проекта — создать простое оконное приложение «Калькулятор», которое:

- принимает два числа;
- выполняет арифметическую операцию;
- отображает результат.

В процессе разработки будут использованы:

- окно;
- поля ввода;
- кнопки;
- обработка событий;
- процедуры.

Проект реализуется средствами языка **PureBasic**.

20.2 Проектирование интерфейса

Программа будет содержать:

- два поля ввода для чисел;
- четыре кнопки (+, −, ×, ÷);
- поле для отображения результата.

Рекомендуемая схема расположения:

[число 1]

[число 2]

[+] [-] [*] [/]

Результат: _____

20.3 Подготовка идентификаторов

```
Enumeration
  #Window_Main
   #String_Number1
  #String_Number2
  #String_Result

  #Button_Add
  #Button_Sub
  #Button_Mul
  #Button_Div
EndEnumeration
```

20.4 Создание интерфейса

```
If OpenWindow(#Window_Main, #PB_Ignore, #PB_Ignore, 300, 250,
          "Калькулятор",
          #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

 TextGadget(#PB_Any, 20, 20, 100, 20, "Число 1:")
 StringGadget(#String_Number1, 20, 40, 260, 25, "")

 TextGadget(#PB_Any, 20, 75, 100, 20, "Число 2:")
 StringGadget(#String_Number2, 20, 95, 260, 25, "")

 ButtonGadget(#Button_Add, 20, 130, 50, 30, "+")
 ButtonGadget(#Button_Sub, 80, 130, 50, 30, "-")
 ButtonGadget(#Button_Mul, 140, 130, 50, 30, "*")
 ButtonGadget(#Button_Div, 200, 130, 50, 30, "/")

 TextGadget(#PB_Any, 20, 175, 100, 20, "Результат:")
 StringGadget(#String_Result, 20, 195, 260, 25, "")
```

20.5 Процедура вычисления

```
Procedure Calculate(operation.s)
  a.d = ValD(GetGadgetText(#String_Number1))
  b.d = ValD(GetGadgetText(#String_Number2))
 Select operation
   Case "+"
     result = a + b
   Case "-"
     result = a - b
   Case "*"
     result = a *
   Case "/"
     If b <> 0
       result = a / b
     Else
       MessageRequester("Ошибка", "Деление на ноль невозможно")
       ProcedureReturn
     EndIf
  EndSelect
  SetGadgetText(#String_Result, StrD(result))
EndProcedure
```

20.6 Обработка событий

```
Repeat
  event = WaitWindowEvent()

  Select event

    Case #PB_Event_Gadget

      Select EventGadget()

        Case #Button_Add
          Calculate("+")

        Case #Button_Sub
          Calculate("-")

        Case #Button_Mul
          Calculate("*")

        Case #Button_Div
          Calculate("/")

      EndSelect

    Case #PB_Event_CloseWindow
      Break

  EndSelect

 ForEver

EndIf
```

20.7 Полный листинг программы

Объедините все части в один файл.
Программа готова к компиляции.

20.8 Улучшение проекта

Возможные доработки:

- проверка корректности ввода;
- добавление кнопки очистки;
- автоматический расчёт при нажатии Enter;
- изменение цвета результата;
- добавление истории вычислений.

20.9 Анализ проекта

В данном проекте были использованы:

- окно;
- элементы управления;
- перечисления;
- процедуры;
- преобразование типов;
- обработка событий;
- проверка деления на ноль.

Практическое задание

1. Добавьте кнопку «Очистить».
2. Реализуйте изменение цвета поля результата.
3. Добавьте поддержку клавиши Enter.
4. Добавьте вычисление процента.

Итоги главы

В данной главе был создан полноценный графический калькулятор.

Это первый завершённый проект, объединяющий основные знания языка.

В следующей главе начнётся итоговый проект — создание полноценного настольного приложения с сохранением данных.

Глава 21

Итоговый проект: Менеджер заметок

21.1 Цель проекта

В рамках итогового проекта будет создано полноценное настольное приложение «Менеджер заметок».

Программа позволит:

- создавать заметки;
- сохранять их в файл;
- загружать при запуске;
- удалять выбранные записи.

Проект объединяет все ранее изученные темы:

- окна и элементы управления;
- меню;
- списки;
- работу со строками;
- работу с файлами;
- обработку событий.

Реализация выполняется средствами языка
PureBasic.

21.2 Проектирование интерфейса

Программа будет содержать:

- поле ввода для новой заметки;
- кнопку «Добавить»;
- список заметок;
- кнопку «Удалить»;

- меню «Файл → Выход».

Структура окна:

[Текст заметки]
[Добавить]

[Список заметок]
[Удалить выбранную]
Меню: Файл → Выход

21.3 Подготовка идентификаторов

```
Enumeration
  #Window_Main

  #Menu_Exit

  #String_Input
  #Button_Add
  #List_Notes
  #Button_Delete
EndEnumeration
```

21.4 Создание интерфейса

```
If OpenWindow(#Window_Main, #PB_Ignore, #PB_Ignore, 400, 400,
              "Менеджер заметок",
              #PB_Window_SystemMenu | #PB_Window_ScreenCentered)

; Меню
If CreateMenu(0, WindowID(#Window_Main))
  MenuTitle("Файл")
  MenuItem(#Menu_Exit, "Выход")
EndIf

; Поле ввода
StringGadget(#String_Input, 20, 20, 360, 25, "")
ButtonGadget(#Button_Add, 20, 55, 100, 30, "Добавить")

; Список
ListViewGadget(#List_Notes, 20, 100, 360, 200)

ButtonGadget(#Button_Delete, 20, 320, 150, 30, "Удалить выбранную")
```

21.5 Работа со списком заметок

Для хранения заметок используем список:

NewList notes.s()

21.6 Добавление заметки

```
Procedure AddNote()

  text.s = GetGadgetText(#String_Input)
  If Trim(text) <> ""
    AddElement(notes())
    notes() = text

    AddGadgetItem(#List_Notes, -1, text)
    SetGadgetText(#String_Input, "")

  EndIf
EndProcedure
```

21.7 Удаление заметки

```
Procedure DeleteNote()
   selected = GetGadgetState(#List_Notes)

  If selected >= 0
      RemoveGadgetItem(#List_Notes, selected)

    SelectElement(notes(), selected)
    DeleteElement(notes())
 EndIf

EndProcedure
```

21.8 Сохранение заметок в файл

```
Procedure SaveNotes()

  If CreateFile(0, "notes.txt")

    ForEach notes()
      WriteStringN(0, notes())
    Next
    CloseFile(0)
  EndIf

EndProcedure
```

21.9 Загрузка заметок при запуске

```
Procedure LoadNotes()
  If ReadFile(0, "notes.txt")
    While Eof(0) = 0
      text.s = ReadString(0)

      AddElement(notes())
      notes() = text
      AddGadgetItem(#List_Notes, -1, text)

    Wend
    CloseFile(0)

  EndIf
EndProcedure
```

После создания интерфейса необходимо вызвать:

LoadNotes()

21.10 Обработка событий

```
Repeat
  event = WaitWindowEvent()
  Select event

    Case #PB_Event_Gadget
      Select EventGadget()
        Case #Button_Add
          AddNote()
        Case #Button_Delete
          DeleteNote()
      EndSelect

    Case #PB_Event_Menu
      If EventMenu() = #Menu_Exit
        SaveNotes()
        Break
      EndIf

    Case #PB_Event_CloseWindow
      SaveNotes()
      Break
  EndSelect

  ForEver

EndIf
```

21.11 Завершение проекта

Программа:

- сохраняет данные при закрытии;
- загружает их при запуске;
- позволяет добавлять и удалять заметки;
- использует список и файл хранения.

Это полноценное настольное приложение.

21.12 Возможные улучшения

1. Добавление редактирования заметки.
2. Подтверждение перед удалением.
3. Сортировка заметок.
4. Выбор папки хранения файла.
5. Использование структуры вместо простой строки.

Итоги проекта

В рамках итогового проекта были применены:

- списки;
- процедуры;
- меню;
- события;
- работа с файлами;
- элементы управления;
- структурирование программы.

На этом базовый курс PureBasic завершён.

Далее можно перейти к:

- работе с модулями;
- многопоточности;
- работе с сетью;
- созданию более сложных приложений.

Глава 22

Модули в PureBasic

22.1 Назначение модулей

По мере роста программы увеличивается объём кода.
Чтобы проект оставался понятным и структурированным, код необходимо разделять на логические части.

Для этого используются модули.

В языке **PureBasic** модуль позволяет:

- группировать связанные процедуры;
- скрывать внутреннюю реализацию;
- избегать конфликтов имён;
- создавать повторно используемые компоненты.

22.2 Общая структура модуля

Модуль состоит из двух частей:

1. DeclareModule — объявление (интерфейс);
2. Module — реализация.

22.3 Простейший пример модуля

```
DeclareModule MathTools

  Declare.i Add(a.i, b.i)
  Declare.i Multiply(a.i, b.i)

EndDeclareModule


Module MathTools

  Procedure.i Add(a.i, b.i)
```

```
  ProcedureReturn a + b
EndProcedure

Procedure.i Multiply(a.i, b.i)
  ProcedureReturn a * b
EndProcedure

EndModule
```
Использование:

```
result = MathTools::Add(5, 3)
MessageRequester("Результат", Str(result))
```

22.4 Объяснение структуры

DeclareModule

Здесь объявляются:

- процедуры;
- структуры;
- константы.

Это «публичный интерфейс» модуля.

Module

Содержит реальную реализацию объявленных процедур.

Всё, что не объявлено в DeclareModule, считается внутренним и недоступным извне.

22.5 Пространство имён

Обращение к процедуре модуля выполняется через:

```
ИмяМодуля::ИмяПроцедуры()
```
Пример:

```
MathTools::Multiply(4, 6)
```
Это предотвращает конфликт имён в крупных проектах.

22.6 Скрытые процедуры

Можно создавать внутренние процедуры, которые не объявлены в DeclareModule.

```
Module Example

  Procedure InternalProcedure()
    Debug "Внутренняя процедура"
  EndProcedure

EndModule
```

Такая процедура доступна только внутри модуля.

22.7 Переменные внутри модуля

Переменные, объявленные внутри Module, становятся локальными для него.

```
Module CounterModule

  Global counter.i

  Procedure Increment()
    counter + 1
    ProcedureReturn counter
  EndProcedure

EndModule
```

Извне переменная counter недоступна напрямую.

22.8 Использование модуля в отдельном файле

В крупных проектах модуль можно вынести в отдельный файл:

MathTools.pb

И подключить его:

XIncludeFile "MathTools.pb"
Это позволяет организовать проект по файлам.

22.9 Пример практического модуля

Создадим модуль для работы с текстом.

```
DeclareModule TextUtils

  Declare.s ToUpper(text.s)
  Declare.s Reverse(text.s)

EndDeclareModule
Module TextUtils

  Procedure.s ToUpper(text.s)
    ProcedureReturn UCase(text)
  EndProcedure

  Procedure.s Reverse(text.s)
    Protected result.s
    For i = Len(text) To 1 Step -1
      result + Mid(text, i, 1)
    Next
    ProcedureReturn result
  EndProcedure

EndModule
```

Использование:

MessageRequester("Тест", TextUtils::Reverse("PureBasic"))

22.10 Когда использовать модули

Модули рекомендуется применять:

- в проектах более 300–500 строк;

- при повторном использовании кода;
- при создании библиотек;
- для логического разделения проекта.

22.11 Типичные ошибки

1. Отсутствие объявления процедуры в DeclareModule.
2. Попытка вызвать внутреннюю процедуру извне.
3. Несоответствие сигнатуры процедуры.
4. Дублирование имён модулей.

Практическое задание

1. Создайте модуль Calculator:
    - сложение;
    - вычитание;
    - умножение;
    - деление.
2. Подключите его в основной программе.
3. Добавьте внутреннюю вспомогательную процедуру.
4. Вынесите модуль в отдельный файл.

Итоги главы

В данной главе были рассмотрены:

- назначение модулей;
- объявление и реализация;
- пространство имён;
- инкапсуляция;
- подключение модулей из файла.

Глава 23

Работа с памятью

23.1 Зачем изучать работу с памятью

Большинство задач в **PureBasic** можно решать без прямого управления памятью.
Однако для создания:

- высокопроизводительных приложений;
- собственных структур данных;
- работы с API операционной системы;
- обработки бинарных данных;

необходимо понимать, как работает память.

23.2 Понятие памяти в программе

Во время работы программа использует оперативную память (RAM) для хранения:

- переменных;
- массивов;
- строк;
- структур;
- временных данных.

Обычно управление памятью происходит автоматически.
Но PureBasic предоставляет инструменты для ручного управления.

23.3 Выделение памяти

Для выделения блока памяти используется функция:

AllocateMemory(размер)
Пример

*buffer = AllocateMemory(100)

- Выделяется 100 байт памяти.
- *buffer — указатель на начало блока.

Важно: переменная с * — это указатель.

23.4 Освобождение памяти

После использования память необходимо освободить:

FreeMemory(*buffer)
Если не освобождать память, возникает **утечка памяти**.

23.5 Запись данных в память

Пример записи числа

```
*memory = AllocateMemory(4)

PokeI(*memory, 12345)

value = PeekI(*memory)

MessageRequester("Значение", Str(value))

FreeMemory(*memory)
```

23.6 Функции Peek и Poke

Функция	Назначение
PokeB()	запись байта
PokeW()	запись 2 байт
PokeL()	запись 4 байт
PokeI()	запись целого
PeekB()	чтение байта
PeekI()	чтение целого

23.7 Работа со строками в памяти

```
text.s = "PureBasic"

*mem = AllocateMemory(StringByteLength(text) + SizeOf(Character))

PokeS(*mem, text)

result.s = PeekS(*mem)

MessageRequester("Строка", result)

FreeMemory(*mem)
```

23.8 Структуры в памяти

```
Structure Point
  x.i
  y.i
EndStructure

*point.Point = AllocateMemory(SizeOf(Point))

*point\x = 10
*point\y = 20

MessageRequester("Координаты",
  "X: " + Str(*point\x) + #CRLF$ +
  "Y: " + Str(*point\y))

FreeMemory(*point)
```

23.9 Работа с указателями

Указатель — это переменная, хранящая адрес в памяти.

Объявление:

```
*ptr.Integer
```

Пример:

```
value.i = 100
*ptr = @value

MessageRequester("Адрес", Str(*ptr\i))
```
Символ @ возвращает адрес переменной.

23.10 Изменение значения через указатель

```
value.i = 50
*ptr = @value

*ptr\i = 200

MessageRequester("Результат", Str(value))
```
Значение изменится через указатель.

23.11 Бинарные данные

Работа с памятью особенно полезна при:

- чтении бинарных файлов;
- обработке изображений;
- сетевых протоколах;
- создании собственных форматов данных.

23.12 Опасности ручного управления памятью

1. Утечки памяти (не освобождённая память).
2. Обращение к освобождённому блоку.
3. Запись за пределами выделенного размера.
4. Неверный тип при использовании Peek/Poke.

Такие ошибки могут привести к сбою программы.

23.13 Когда использовать ручное управление памятью

Рекомендуется применять:

- при работе с системными API;
- при создании высокопроизводительных структур;
- при обработке больших объёмов данных;
- в продвинутых проектах.

В обычных приложениях чаще достаточно стандартных средств языка.

Практическое задание

1. Выделите память для массива из 10 целых чисел.
2. Запишите значения от 1 до 10.
3. Считайте их и выведите сумму.
4. Освободите память.
5. Создайте структуру в памяти и измените её поля через указатель.

Итоги главы

В данной главе были рассмотрены:

- выделение и освобождение памяти;
- указатели;
- функции Peek и Poke;
- структуры в памяти;
- риски ручного управления памятью.

Работа с памятью — это переход от базового к системному уровню программирования.

Глава 24

Создание собственных библиотек

24.1 Зачем создавать библиотеки

По мере роста проекта часто возникает необходимость:

- повторно использовать код в разных программах;
- распространять собственные инструменты;
- скрывать реализацию;
- создавать расширяемую архитектуру.

Для этого создаются библиотеки.

В языке **PureBasic** библиотека может быть реализована двумя способами:

1. как модуль, подключаемый через `XIncludeFile`;
2. как динамическая библиотека (DLL).

В данной главе будут рассмотрены оба подхода.

Часть I

Библиотека на основе модуля

24.2 Создание библиотечного файла

Создадим файл:

`MyLibrary.pb`
Содержимое:

```
DeclareModule MyLibrary

  Declare.s GetVersion()
  Declare.i Square(value.i)

EndDeclareModule
```

```
Module MyLibrary

  Procedure.s GetVersion()
    ProcedureReturn "1.0"
  EndProcedure

  Procedure.i Square(value.i)
    ProcedureReturn value * value
  EndProcedure

EndModule
```

24.3 Подключение библиотеки

В основной программе:

```
XIncludeFile "MyLibrary.pb"

MessageRequester("Версия", MyLibrary::GetVersion())
MessageRequester("Результат", Str(MyLibrary::Square(5)))
```

24.4 Преимущества такого подхода

- простота реализации;
- удобство редактирования;
- кроссплатформенность;
- не требует компиляции отдельного файла.

Недостаток: код доступен в открытом виде.

Часть II

Создание DLL-библиотеки

24.5 Что такое DLL

DLL (Dynamic Link Library) — это отдельный файл с расширением .dll, содержащий функции, которые можно использовать в других программах.

DLL:

- компилируется отдельно;
- может использоваться разными приложениями;
- скрывает исходный код.

24.6 Создание DLL в PureBasic

Создадим новый файл:

MyDLL.pb
Пример содержимого:

```
ProcedureDLL.i Add(a.i, b.i)
  ProcedureReturn a + b
EndProcedure

ProcedureDLL.s GetMessage()
  ProcedureReturn "Привет из DLL"
EndProcedure
```

24.7 Компиляция DLL

В настройках компилятора необходимо выбрать:

Create Executable → Shared DLL
После компиляции будет создан файл:

MyDLL.dll

24.8 Подключение DLL в программе

В основной программе:

```
Import "MyDLL.dll"
  Add(a.i, b.i)
  GetMessage()
EndImport
```

```
result = Add(5, 3)
MessageRequester("Сумма", Str(result))

MessageRequester("Сообщение", GetMessage())
```

24.9 Передача строк через DLL

При работе со строками рекомендуется учитывать формат (Unicode / ASCII).
Чаще всего используется стандартная строковая модель PureBasic.

24.10 Экспорт структур через DLL

Можно экспортировать процедуры, работающие со структурами:

```
Structure Point
  x.i
  y.i
EndStructure

ProcedureDLL MovePoint(*p.Point)
  *p\x + 10
  *p\y + 10
EndProcedure
```

24.11 Когда использовать DLL

DLL рекомендуется использовать:

- при создании закрытых коммерческих компонентов;
- при разработке плагинов;
- при интеграции с другими языками;
- при создании расширяемых систем.

24.12 Рекомендации по архитектуре библиотеки

1. Использовать модули внутри DLL.
2. Минимизировать глобальные переменные.
3. Чётко документировать экспортируемые функции.
4. Сохранять совместимость версий.

5. Использовать единый стиль именования.

24.13 Типичные ошибки

1. Несоответствие сигнатуры в Import.
2. Ошибка разрядности (32/64 бит).
3. Проблемы с передачей строк.
4. Отсутствие DLL в папке запуска программы.

Практическое задание

1. Создайте DLL с функциями:
    - умножение;
    - деление;
    - вычисление степени числа.
2. Подключите DLL в отдельной программе.
3. Добавьте процедуру, работающую со структурой.
4. Создайте версию 2.0 библиотеки.

Итоги главы

В данной главе были рассмотрены:

- создание библиотеки через модуль;
- подключение через XIncludeFile;
- создание DLL;
- экспорт функций;
- импорт в программе;
- рекомендации по архитектуре.

Создание библиотек — это шаг к профессиональной разработке.

Глава 25

Работа с Windows API в PureBasic

25.1 Что такое Windows API

Windows API (Application Programming Interface) — это набор функций операционной системы Windows, позволяющих:

- создавать окна и управлять ими;
- работать с файлами и процессами;
- управлять памятью;
- взаимодействовать с оборудованием;
- работать с сетью и системой.

PureBasic уже использует системные функции Windows, но при необходимости программист может обращаться к ним напрямую.

Работа с API позволяет:

- расширить возможности программы;
- получить доступ к низкоуровневым функциям;
- реализовать нестандартное поведение интерфейса.

25.2 Использование API в языке PureBasic

PureBasic предоставляет встроенную поддержку вызова Windows API.

Чаще всего используются:

- Import
- ImportC
- встроенные объявления структур Windows
- встроенные константы

25.3 Пример: изменение заголовка окна через API

PureBasic создаёт окно, но можно изменить его заголовок напрямую через Windows API.

```
OpenWindow(0, 100, 100, 400, 200, "Оригинальный заголовок")

SetWindowText_(WindowID(0), "Новый заголовок через API")

Repeat
  event = WaitWindowEvent()
Until event = #PB_Event_CloseWindow
```

Обратите внимание на символ _ в конце имени функции.
PureBasic автоматически предоставляет доступ к API-функциям Windows с добавлением подчёркивания.

25.4 Пример: получение дескриптора окна

```
hwnd = WindowID(0)
MessageRequester("HWND", Str(hwnd))
```

HWND — это уникальный идентификатор окна в системе Windows.

25.5 Пример: вывод стандартного системного окна MessageBox

```
MessageBox_(0, "Сообщение из Windows API",
          "API", #MB_OK | #MB_ICONINFORMATION)
```

Это вызов функции Windows напрямую, без использования MessageRequester().

25.6 Работа со структурой Windows

Многие функции Windows API используют структуры.

Пример: получение информации о системе.

```
Structure SYSTEMTIME
  wYear.w
  wMonth.w
  wDayOfWeek.w
  wDay.w
  wHour.w
  wMinute.w
  wSecond.w
  wMilliseconds.w
EndStructure

Define time.SYSTEMTIME

GetLocalTime_(@time)

MessageRequester("Время",
  Str(time\wHour) + ":" +
  Str(time\wMinute) + ":" +
  Str(time\wSecond))
```

25.7 Пример: изменение позиции окна

```
OpenWindow(0, 100, 100, 400, 200, "Перемещение")
MoveWindow_(WindowID(0), 300, 200, 400, 200, #True)

Repeat
  event = WaitWindowEvent()
Until event = #PB_Event_CloseWindow
```

25.8 Пример: установка прозрачности окна

```
OpenWindow(0, 100, 100, 400, 200, "Прозрачность",
         #PB_Window_SystemMenu)

SetWindowLongPtr_(WindowID(0), #GWL_EXSTYLE,
  GetWindowLongPtr_(WindowID(0), #GWL_EXSTYLE) | #WS_EX_LAYERED)

SetLayeredWindowAttributes_(WindowID(0), 0, 180, #LWA_ALPHA)

Repeat
  event = WaitWindowEvent()
Until event = #PB_Event_CloseWindow
```

Значение 180 определяет степень прозрачности (0–255).

25.9 Импорт функций вручную

Если функция не объявлена автоматически, её можно импортировать:

```
Import "user32.lib"
  MessageBeep(dwType.l)
EndImport

MessageBeep(#MB_ICONINFORMATION)
```

25.10 Основные типы Windows

Тип	Назначение
HWND	дескриптор окна
HANDLE	универсальный дескриптор
DWORD	32-битное число
LPARAM	параметр сообщения
WPARAM	параметр сообщения

25.11 Когда использовать Windows API

Работа с API необходима:

- при создании нестандартного интерфейса;
- при управлении стилями окна;
- при создании глобальных горячих клавиш;
- при работе с системными процессами;
- при интеграции с Windows-службами.

25.12 Риски при работе с API

1. Ошибка в типах параметров.
2. Неправильная структура.
3. Нарушение совместимости 32/64 бит.

4. Потеря кроссплатформенности (API Windows работает только в Windows).

25.13 Практическое задание

1. Создайте окно.
2. Измените его прозрачность.
3. Выведите системное сообщение через MessageBox_.
4. Получите текущее время через Windows API.
5. Попробуйте переместить окно через MoveWindow_.

Итоги главы

В данной главе были рассмотрены:

- основы Windows API;
- вызов системных функций;
- работа со структурами Windows;
- изменение параметров окна;
- импорт функций вручную.

Глава 26

Создание собственного GUI-фреймворка в PureBasic

26.1 Зачем создавать свой GUI-фреймворк

Стандартные средства **PureBasic** позволяют быстро создавать интерфейс.
Однако в крупных проектах возникает необходимость:

- единообразного стиля;
- централизованной обработки событий;
- повторно используемых компонентов;
- собственной системы тем оформления;

```
Module UI

  Structure CallbackEntry
    id.i
    *proc
  EndStructure

  Global NewList callbacks.CallbackEntry()
  Global mainWindow.i

  Procedure Init()
    ; Инициализация при необходимости
  EndProcedure

  Procedure CreateMainWindow(title.s, width.i, height.i)
    mainWindow = OpenWindow(0, #PB_Ignore, #PB_Ignore, width, height,
                            title,
                            #PB_Window_SystemMenu | #PB_Window_ScreenCentered)
  EndProcedure

  Procedure AddButton(id.i, x.i, y.i, w.i, h.i, text.s)
    ButtonGadget(id, x, y, w, h, text)
  EndProcedure

  Procedure SetCallback(id.i, *callback)
    AddElement(callbacks())
    callbacks()\id = id
    callbacks()\proc = *callback
  EndProcedure

  Procedure Run()
    Repeat
      event = WaitWindowEvent()

      Select event
        Case #PB_Event_Gadget

          ForEach callbacks()
            If callbacks()\id = EventGadget()
              CallFunctionFast(callbacks()\proc)
            EndIf
          Next
        Case #PB_Event_CloseWindow
          Break
      EndSelect

    ForEver
  EndProcedure

EndModule
```

- архитектуры «окно → контролы → логика».

GUI-фреймворк — это надстройка над стандартными средствами создания окон, упрощающая разработку.

26.2 Архитектурная идея

Будем строить простой фреймворк со следующими принципами:

1. Центральный модуль UI
2. Регистрация окон
3. Регистрация элементов
4. Централизованный цикл событий
5. Поддержка callback-процедур

26.3 Базовая структура фреймворка

Создадим файл:

UIFramework.pb

26.4 Интерфейс модуля

```
DeclareModule UI

  Declare Init()
  Declare CreateMainWindow(title.s, width.i, height.i)
  Declare AddButton(id.i, x.i, y.i, w.i, h.i, text.s)
  Declare SetCallback(id.i, *callback)
  Declare Run()

EndDeclareModule
```

26.5 Реализация модуля

26.6 Использование фреймворка

Создадим основной файл программы:

```
XIncludeFile "UIFramework.pb"

Procedure OnButtonClick()
  MessageRequester("Событие", "Кнопка нажата")
EndProcedure

UI::Init()
UI::CreateMainWindow("Мой фреймворк", 400, 200)

UI::AddButton(1, 150, 80, 100, 30, "Нажать")
UI::SetCallback(1, @OnButtonClick())

UI::Run()
```

26.7 Что мы реализовали

Фреймворк теперь:

- создаёт окно;
- создаёт кнопки;
- регистрирует callback;
- централизованно обрабатывает события;
- отделяет интерфейс от логики.

Это уже архитектурный уровень разработки.

26.8 Расширение фреймворка

Можно добавить:

1. Систему тем

```
Global themeColor.i
```

И применять цвет ко всем элементам.

2. Автоматическую регистрацию элементов

Хранить список всех созданных контролов.

3. Поддержку нескольких окон

Добавить структуру WindowEntry и управлять ими централизованно.

4. Абстракцию элементов

Создать универсальную структуру:

```
Structure UIElement
  id.i
  type.i
  x.i
  y.i
  width.i
  height.i
EndStructure
```

5. Систему layout

Автоматическое позиционирование элементов.

26.9 Преимущества собственного фреймворка

- повторное использование;
- единая архитектура;
- централизованный контроль;
- расширяемость;
- профессиональный уровень разработки.

26.10 Ограничения

- увеличение сложности;
- необходимость тестирования;

- риск переусложнения;
- поддержка кроссплатформенности.

26.11 Когда имеет смысл создавать свой фреймворк

- проект более 2000 строк;
- разрабатывается несколько приложений;
- требуется единый стиль;
- создаётся коммерческий продукт.

26.12 Практическое задание

1. Добавьте поддержку StringGadget.
2. Реализуйте callback с передачей параметра.
3. Добавьте поддержку меню.
4. Добавьте систему тем (цвет фона окна).
5. Создайте второй пример приложения на основе фреймворка.

Итоги главы

В данной главе были рассмотрены:

- архитектурный подход к GUI;
- создание модуля-фреймворка;
- регистрация callback;
- централизованный цикл событий;
- расширяемость системы.

Вы перешли от простого написания программ к созданию собственной архитектуры.

Заключение издания

Вы прошли путь от первых строк кода до построения собственной архитектуры приложений.

Освоены:

- базовые конструкции языка;
- структуры данных;
- работа с файлами;
- создание GUI;
- модульная организация кода;
- работа с памятью;
- системное программирование;
- разработка библиотек и фреймворков.

Курс завершён.