Введение в ООП

Характеристики языков программирования

Мощность, уровень и концептуальная целостность

Мощность - количество и разнообразие задач, алгоритмы, решения которых можно записать

Уровень языка - сложность решения задач с помощью этого языка

Концептуальная целостность языка - экономия, ортогональность и единообразие понятий.

Экономия понятий - достижение максимальной мощности языка с помощью минимального числа понятий.
Ортогональность понятий - между понятиями не должно быть взаимного влияния (правила использования должны быть одни и те же).

Единообразие понятий - согласованный, единого подхода к описанию и использованию всех понятий.

Связь мощности языка с его уровнем и концептуальной ценностью?

Свойства языков программирования

Надежность, удобочитаемость, полнота, гибкость, простота

Надежность языка обеспечивает минимум ошибок при написании программ.

Удобочитаемость языка — это свойство, обеспечивающее легкость восприятия программ человеком.

Полнота языка обеспечивает описание на языке решения задач определенной предметной области, а также с помощью средств языка, например средств отладки, процесса разработки программ

Гибкость языка обеспечивает легкость выражения на языке необходимых для решения задач действий, предоставляет программисту достаточно возможностей для выражения всех операций в программе.

Простота языка обеспечивает легкость понимания семантики языковых конструкций и запоминания их синтаксиса.

Мобильность языка обеспечивает независимость его аппаратных средств, позволяет переносить программное обеспечение с машины на машину с относительной легкостью

Эффективность языка обеспечивает эффективную реализацию языка.

Язык программирования позволяет минимизировать суммарное время и энергию, затрачиваемые на решение задач на ЭВМ

Приведите пример!

Предпосылки возникновения ООП

1. Эволюция типов данных

Простые типы

Составные типы

Типы, определенные пользователем

2. Эволюция парадигм программирования

Машинное кодирование

Последовательное программирование (процедурное программирование)

Логическое программирование

Структурное программирование

Функциональное программирование

Модульное программирование

Тенденция к структуризации путем все более тесного объединения взаимосвязанных данных и фрагментов кода в одном блоке при все большем отделении таких блоков друг от друга.

```
ADR  CODE           LINE      COMMENT
                    1     ; вычисление факториала
A000 DB             2     ; значение N
                    3     ;
4000 8B 06 00 A0    4     ; загрузка N в аккумулятор
4004 3D 01 00       5     ; if (@N=1) or (@N=0) then
4007 77 00 41       6     ; переход на вычисление
400A B8 01 00       7     ; factorial:=1
400D 33 12          8     ; очистка регистра
400F C3             9     ; возврат
                    10    ; Вычисление
4100 48             11    ; else
4101 50             12    ; аккумулятор в стек
4102 E8 00 40       13    ; factorial := factorial(@N-1)
4105 F7 E6          14    ; * @N;
4107 C3             15    ; возврат
```

Машинное кодирование

(представление информации на машинных носителях в кодах микропроцессора)

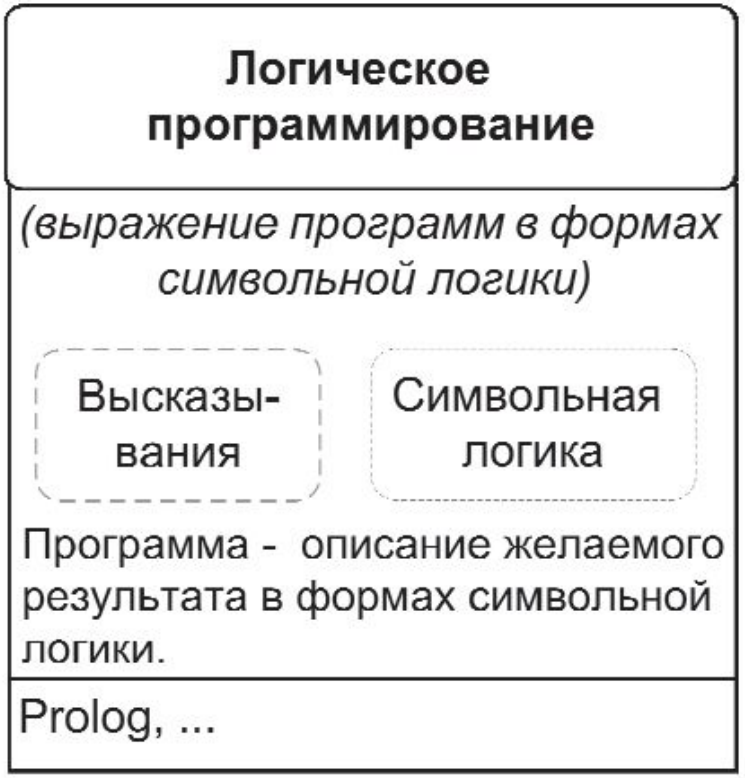
Адрес ячейки и значения

Инструкции вычислительной системы

Программа - список адресов ячеек, их значений и операций над ними.

Шестнадцатеричный код


```
10 n=0
20 j=0
30 k=0
40 cls
50 input "Введите значение n = ", n$
60 if n<1 goto 90
70 k=1
80 for j=1 to n
90 k=k*j
100 next j
110 print " Факториал равен= "; k$
120 END
```




```
domains
        N,F=real
predicates
        factorial(N,F)
        result
clauses
        factorial(0,1).
        factorial(N,F):-N>0,N1=N-1,factorial(N1,F1),
                        F=F1*N.
        result:-write("Input N"),nl,
        write("N="),readreal(N),factorial(N,F),
        write(N,"!=",F).
goal
        result
```


```
(defun fact (n)
       (if (eql n 0)
            1
            (* n (fact (- n 1)))))
(fact 10)
```


```
function fact(n : integer) : longint;
  begin
    if n <= 1 then
      fact := 1
    else
      fact := n * fact(n - 1);
  end;
```

Предпосылки возникновения ООП

- необходимость повышения производительности и разработки за счет многократного (повторного) использования ПО
- необходимость упрощения сопровождения и модификации разработанных систем (локализация вносимых изменений);
- облегчение проектирования систем

Объектная модель является наиболее естественным способом представления реального мира.

Понятие объект-ориентированного программирования

- Термин "объектно-ориентированное программирование" принят преимущественно в российской литературе:

Объектно-ориентированное программирование (ООП, Object-Oriented Programming) - совокупность принципов, технологий, а также инструментальных средств для создания программных систем на основе архитектуры взаимодействия объектов.

- В западной литературе под этим термином понимается сразу три аспекта:

OOP **OOD** **OOA**

OOP (object-oriented programming) – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования

OOD (object-oriented design) – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

OOA (object-oriented analysis) – это методология, при использовании которой требования к проектируемой системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

В основе ООП

- лежит **объектная декомпозиция**, при этом:
- **статическая структура системы описывается в терминах объектов и связей между ними**
- **поведение системы описывается в терминах обмена сообщениями между объектами.**
- Каждый **объект системы обладает своим собственным поведением**, моделирующим поведение объекта

Ключевые положения ООП (Alan Key)

1) Всё является объектом.
2) Вычисления осуществляются путём взаимодействия (обмена данными) между объектами, при котором один объект требует, чтобы другой объект выполнил некоторое действие. Объекты взаимодействуют, посылая и получая сообщения. Сообщение – это запрос на выполнение действия, дополненный набором аргументов, которые могут понадобиться при выполнении действия.
3) Каждый объект имеет независимую память, которая состоит из других объектов.
4) Каждый объект является представителем (экземпляром) класса, который выражает общие свойства объектов.
5) В классе задаётся поведение (функциональность) объекта. Тем самым все объекты, которые являются экземплярами одного класса, могут выполнять одни и те же действия.
6) Классы организованы в единую древовидную структуру с общим корнем, называемую иерархией наследования. Память и поведение, связанное с экземплярами определённого класса, автоматически доступны любому классу, расположенному ниже в иерархическом дереве.

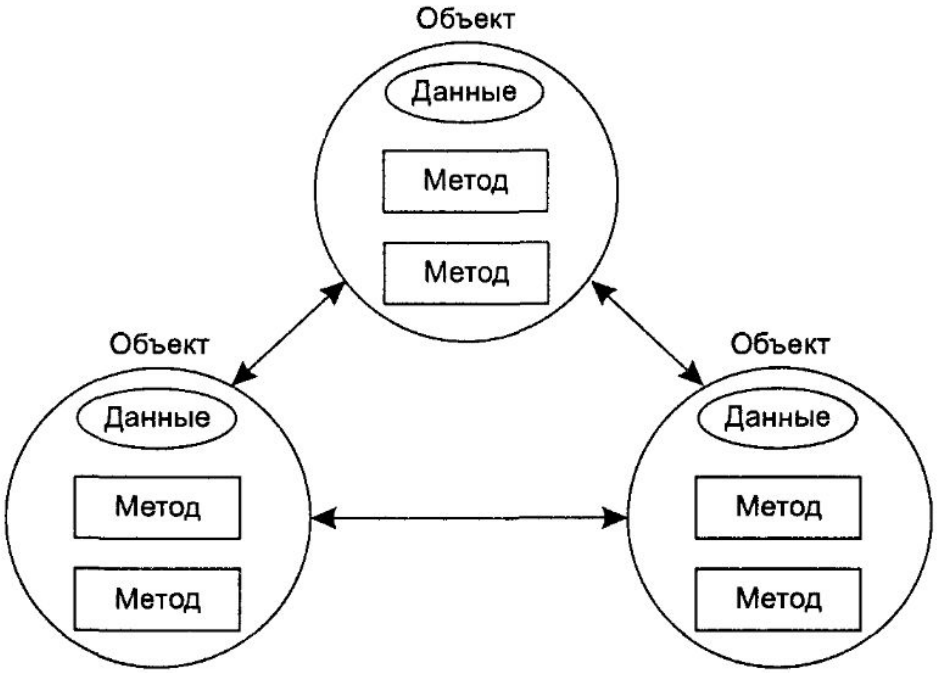
Классификация объектно-ориентированных языков программирования

1) Чистые объектно-ориентированные языки – это те, которые позволяют использовать только одну модель программирования – объектно-ориентированную (Eiffel, Smalltalk).
2) Гибридные языки, которые позволяют программистам использовать при необходимости возможности нескольких парадигм программирования (C++ и Delphi)
3) «Урезанные» языки, которые появились в результате удаления из гибридных языков наиболее опасных и ненужных с объектно-ориентированной точки зрения конструкций (Java, C#).

Классы и объекты

- **Класс –** абстрактное описание свойств и методов для совокупности похожих объектов, представители которой называются экземплярами класса.
- **Объект –** это модель или абстракция реальной сущности в программной системе (экземпляр класса).
- **Свойство объекта –** объявленный в классе параметр, характеризующий объект.
- **Метод объекта –** объявленная в классе процедура, описывающая поведение объекта.

Объекты


Объект
Данные
Метод
Метод
Объект
Данные
Метод
Метод
Объект
Данные
Метод
Метод

Пример


Отдел продаж
Данные о продажах
Менеджер по продажам
Секретарь
Отдел кадров
Данные о работниках
Начальник ОК
Персонал
Финансовый отдел
Данные о финансах
Главный бухгалтер
Бухгалтер

Объекты

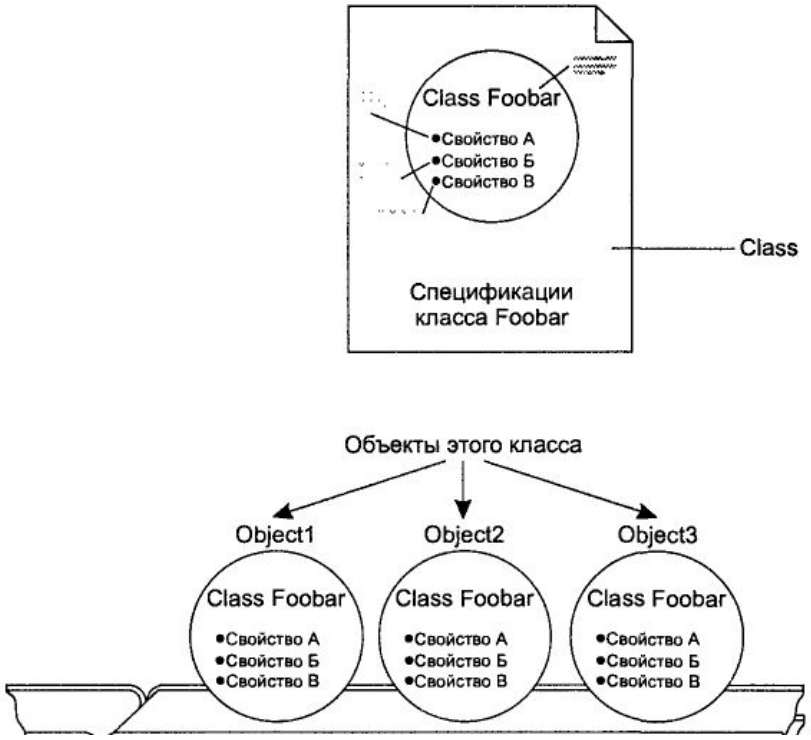
Физические объекты
Автомобили при моделировании уличного движения.
Схемные элементы при моделировании цепи электрического тока.
Страны при создании экономической модели.
Самолеты при моделировании диспетчерской системы.
Элементы интерфейса.
Окна.
Меню.
Графические объекты (линии, прямоугольники, круги).
Мышь, клавиатура, дисковые устройства, принтеры.
Структуры данных.
Массивы.
Связанные списки.
Бинарные деревья.
Группы людей.
Сотрудники.
Студенты.
Покупатели.
Продавцы.

Объекты

Хранилища данных.
Описи инвентаря.
Списки сотрудников.
Словари.
Географические координаты городов мира.
Пользовательские типы данных.
Время.
Величины углов.
Комплексные числа.
Точки на плоскости.
Участники компьютерных игр.
Автомобили в гонках.
Позиции в настольных играх (шашки, шахматы).
Животные в играх, связанных с живой природой.
Друзья и враги в приключенческих играх.

Классы

Класс является описанием совокупности сходных между собой объектов.



Объектная модель

строится на четырех основных принципах (абстрагирование, инкапсуляция, полиморфизм и наследование)

1) **Абстрагирование –** выделение существенных характеристик некоторого объекта, отличающих его от всех других видов объектов

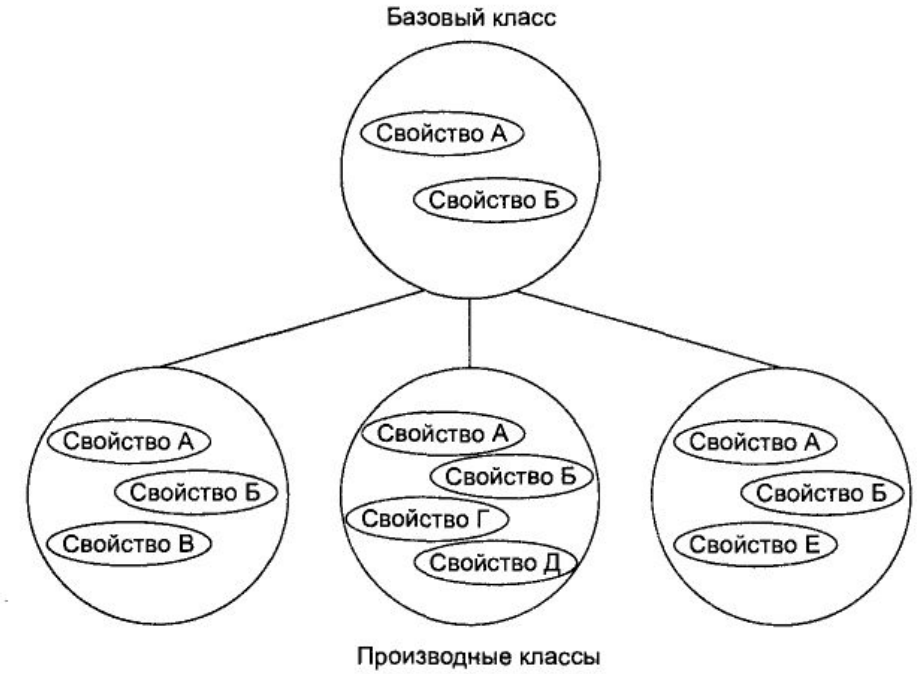
2) **Инкапсуляция –** процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение.

3) **Полиморфизм** – обозначение различных действий одним именем и свойство объекта отвечать на направленный к нему запрос сообразно своему типу (способность функции обрабатывать данные разных типов).

4) **Наследование** – механизм, позволяющий определять новые типы данных на основе существующих таким об разом, что данные и функции существующих классов становятся членами наследуемых классов.

Наследование

- Классы разбиваются на подклассы. В C++ класс, который порождает все остальные классы, называется **базовым классом**, остальные классы наследуют его свойства, одновременно обладая собственными свойствами. Такие классы называются **производными.** Роль наследования в ООП — сократить размер кода и упростить связи между элементами программы.

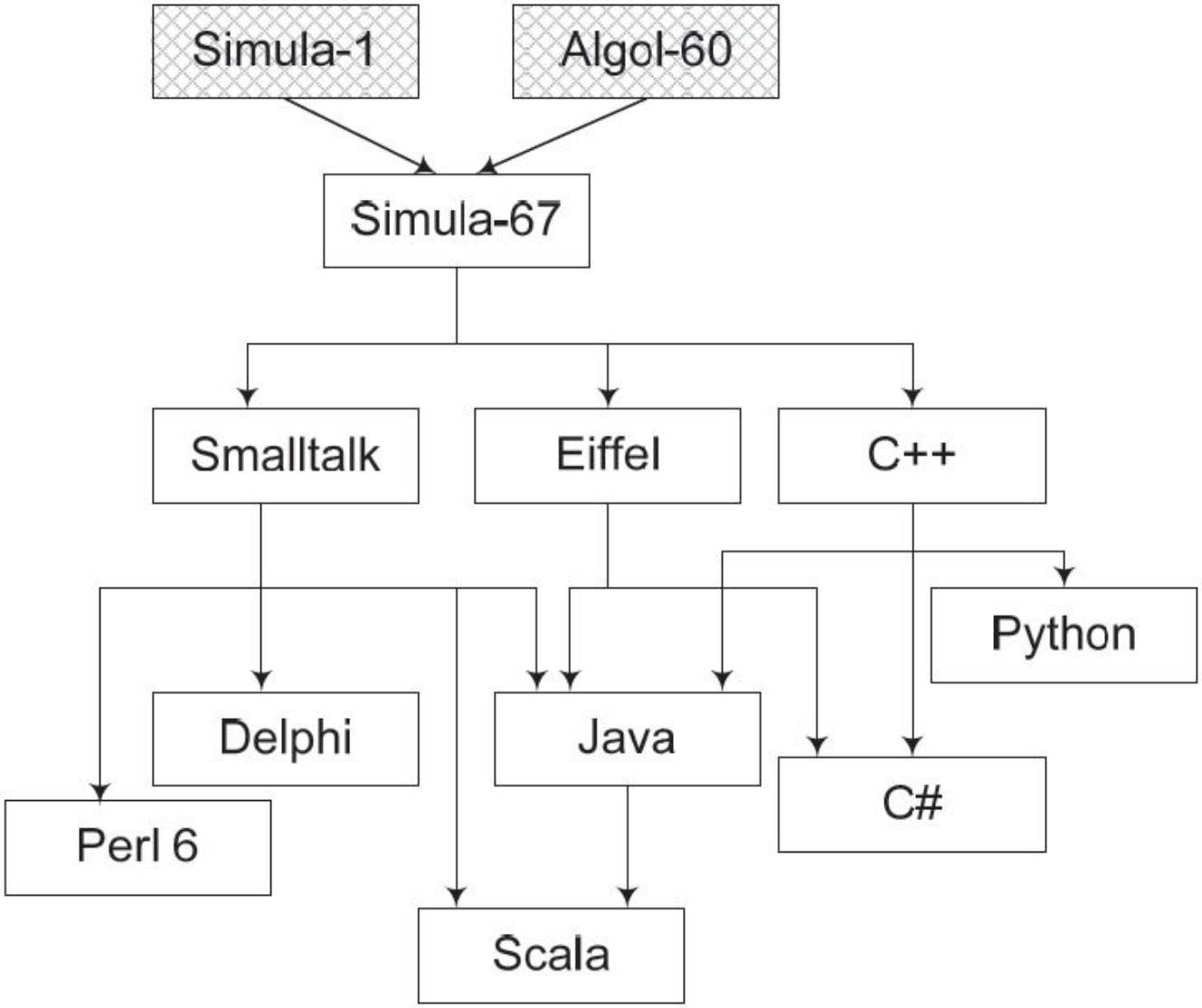


Эволюция объектно-ориентированных языков программирования

Симула-67 (Simula 67) — первый объектно-ориентированный язык программирования. Разработан в конце 60-х сотрудниками Норвежского Вычислительного Центра (Осло) Кристеном Нюгором и Уле-Йоханом Далем для моделирования сложных систем.

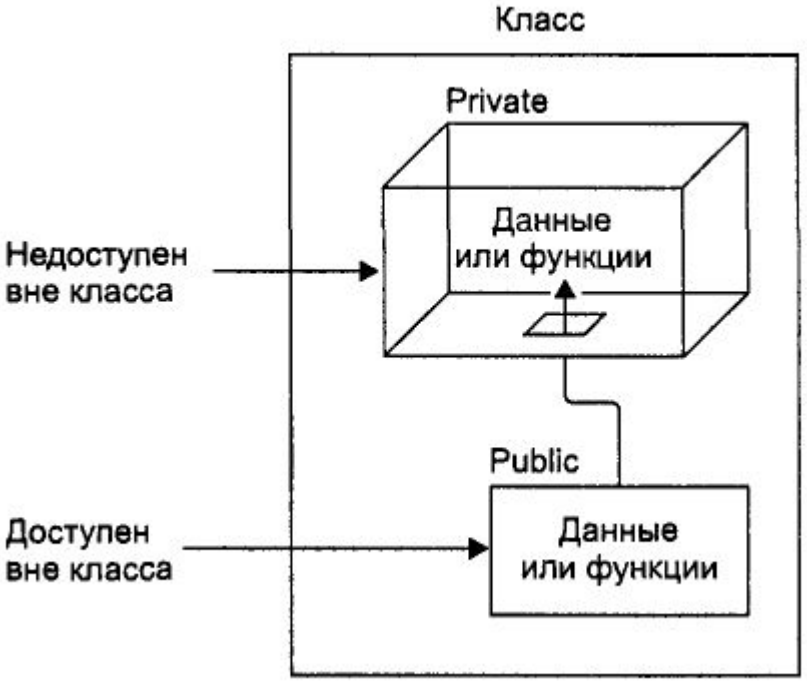
Построен на основе языка Algol-60.

Smalltalk — объектно-ориентированный язык программирования с динамической типизацией, основанный на идее посылки сообщений, разработанный в Xerox PARC Аланом Кэй, Дэном Ингаллсом, Тедом Кэглером, Адель Голдберг, и другими в 1970-х годах.


Simula-1
Algol-60
Simula-67
Smalltalk
Eiffel
C++
Python
Delphi
Java
C#
Perl 6
Scala

private и public

Ключевой особенностью объектно-ориентированного программирования является возможность сокрытия данных. Это означает сокрытие данных одного класса от другого класса.



Простой класс

```
// smallobj.cpp
// демонстрирует простой небольшой объект
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class smallobj                  // определение класса
{
  private:
    int somedata;               // поле класса
  public:
    void setdata(int d)         // метод класса, изменяющий значение поля
        { somedata = d; }
    void showdata()             // метод класса, отображающий значение поля
      { cout << "Значение поля равно " << somedata << endl; }
};
////////////////////////////////////////////////////////////
int main()
{
  smallobj s1, s2;              // определение двух объектов класса smallobj
  s1.setdata(1066);             // вызовы метода setdata()
  s2.setdata(1776);
  s1.showdata();                // вызовы метода showdata()
  s2.showdata();
  return 0;
}
```

Описание класса

```
#include <stdio.h>
class First
{
public:
     char c;
     int x,y;
     /* компонентные функции, определенные внутри класса */
     void print(void)
     {
          printf ("%c %d %d ",c,x,y);
     }
     void set(char ach,int ax,int ay)
     {
          c=ach;     x=ax;     y=ay;
     }
};
```

Описание класса

```
class First
{
  public:   char c;
            int x,y;
   void print(void);
   void set(char ach,int ax,int ay);
};
```

/* компонентные функции, описанные вне класса */

```
void First::print(void)
{   printf ("%c %d %d ",c,x,y);   }
void First::set (char ach,int ax,int ay)
{   c=ach;    x=ax;    y=ay;   }
```

Приемы работы с классами и объектами

Пусть в процессе работы программы создан объект класса

ID класса ID объекта;

и пусть в составе класса есть метод с идентификатором *ID метода.*

Тогда:

1. *Прямой вызов метода*

   ID объекта. ID метода; //- использована операция привязки «точка».

2. *Использование косвенной адресации*

   Пусть объявлен указатель с типом *ID_класса:*

   ID класса *ID указателя;

   Теперь указатель типа *ID класса* можно устанавливать на существующие объекты этого класса:

   ID указателя = & ID объект;

   - указатель идентифицирован адресом объекта.

   Косвенный вызов метода:

   ID указателя -> ID метода;

   - использована операция привязки «стрелка».

3. *Создание объектов* (безымянных) *в динамической области памяти*

   ID класса * ID указателя - new ID класса;

   Компилятор создает объект на этапе работы проекта. Косвенный вызов метода:

   ID_указателя -> ID_метода;

Пример. Сложение двух целых чисел.

```
#include <iostream.h>

class CMySum
{
        int i;
        int j;
        int rezult;
public:
        void Set(int a, int b);
        void Print();
        void CalculateSum();
};
void CMySum::Set(int a, int b)
{
        i = a;
        j = b;
}

void CMySum::Print()
{
 cout << "i="<<i << " j=" <<j << " i+j="<<rezult << "\n";
}

void CMySum::CalculateSum()
{
        rezult = i+j;
}
```

```
void main(void)

{
        CMySum objMySum;
        CMySum *p_objMySum;

        cout << "Direct method.\n";
        objMySum.Set(2,3);
        objMySum.CalculateSum();
        objMySum.Print();

        cout << "Indirect method.\n";
        p_objMySum = &objMySum;
        p_objMySum->Set(4,5);
        p_objMySum->CalculateSum();
        p_objMySum->Print();
}
```

Пример

```
// objpart.cpp
// детали изделия в качестве объектов
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class part                          // определение класса
{
  private:
  int modelnumber;                  // номер изделия
  int partnumber;                   // номер детали
  float cost;                       // стоимость детали
  public:
    // установка данных
    void setpart(int mn, int pn, float c)
   {
      modelnumber = mn;
      partnumber = pn;
      cost = c;
    }
    void showpart()                 // вывод данных
    {
      cout << "Модель " << modelnumber;
      cout << ", деталь " << partnumber;
      cout << ", стоимость $" << cost << endl;
    }
};
////////////////////////////////////////////////////////////
int main()
{
  part part1;                       // определение объекта
                                    // класса part
  part1.setpart(6244, 373, 217.55F); // вызов метода
  part1.showpart();                 // вызов метода
  return 0;
}
```

Пример

```
// circles.cpp
// круги в качестве объектов
#include "msoftcon.h"       // для функций консольной графики
////////////////////////////////////////////////////////////
class circle                //графический объект "круг"
{
  protected:
    int xCo,yCo;            // координаты центра
    int radius;
    color fillcolor;        // цвет
    fstyle fillstyle;       // стиль заполнения
  public:                   // установка атрибутов круга
    void set(int x, int y, int r, color fc, fstyle fs)
    {
      xCo = x;
      yCo = y;
      radius = r;
      fillcolor = fc;
      fillstyle = fs;
    }
    void draw()                     // рисование круга
    {
      set_color(fillcolor);         // установка цвета и
      set_fill_style(fillstyle);    // стиля заполнения
      draw_circle(xCo,yCo,radius); // рисование круга
    }
};
////////////////////////////////////////////////////////////
int main()
{
  init_graphics();          // инициализация графики
  circle c1;                // создание кругов
  circle c2;
  circle c3;
  // установка атрибутов кругов
  c1.set(15, 7, 5, cBLUE, X_FILL);
  c2.set(41, 12, 7, cRED, O_FILL);
  c3.set(65, 18, 4, cGREEN, MEDIUM_FILL);
  c1.draw();                // рисование кругов
  c2.draw();
  c3.draw();
  set_cursor_pos(1,25);     // нижний левый угол
  return 0;
}
```

Класс как тип данных

```
// englobj.cpp
// длины в английской системе в качестве объектов
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class Distance                          // длина в английской системе
{
  private:
    int feet;
    float inches;
  public:
    void setdist( int ft, float in ) // установка значений полей
      { feet = ft; inches = in; }
    void getdist()                      // ввод полей с клавиатуры
    {
      cout << "\nВведите число футов: "; cin >> feet;
      cout << "Введите число дюймов: "; cin >> inches;
    }
    void showdist()                     // вывод полей на экран
      { cout << feet << "\'-" << inches << '\"'; }
};
////////////////////////////////////////////////////////////
int main()
{
  Distance dist1, dist2;                // две длины
  dist1.setdist(11,6.25);               // установка значений для d1
  dist2.getdist();                      // ввод значений для dist2
  // вывод длин на экран
  cout << "\ndist1 = "; dist1.showdist();
  cout << "\ndist2 = "; dist2.showdist();
  cout << endl;
  return 0;
}
```

Организация Windows-программы

- Компьютер
  - Операционная система
    - Программа пользователя

Как они взаимодействуют?

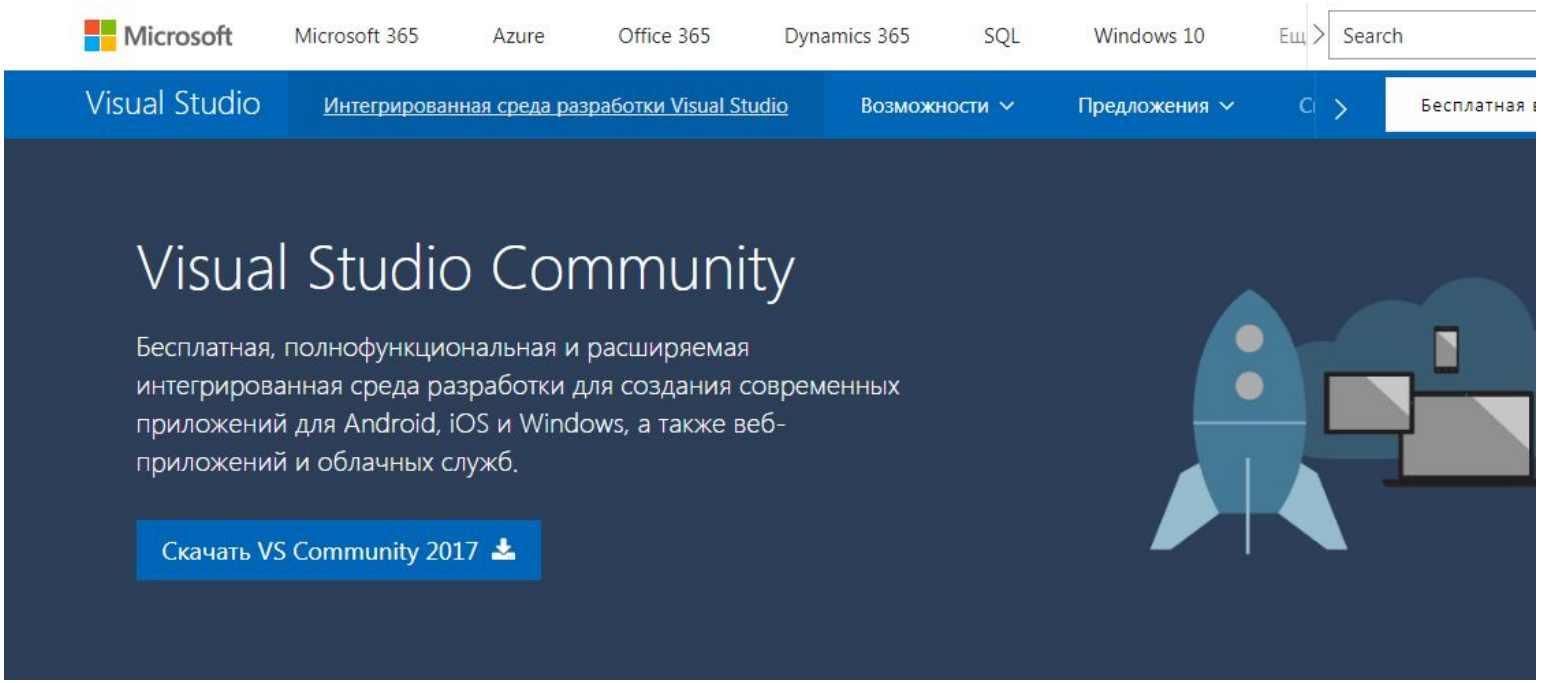
Любое приложение в Windows = Функция инициализации (WinMain) + функция обслуживания приложения (WinProc)

Основное назначение WinMain связано с решением 3 основных задач:

– зарегистрировать в Windows класс главного окна приложения (всех используемых окон)
– создать главное окно и показать его на экране
– организовать цикл обнаружения поступающих в приложение сообщений

Функция обслуживания приложения WinProc – обработка присланных сообщений.

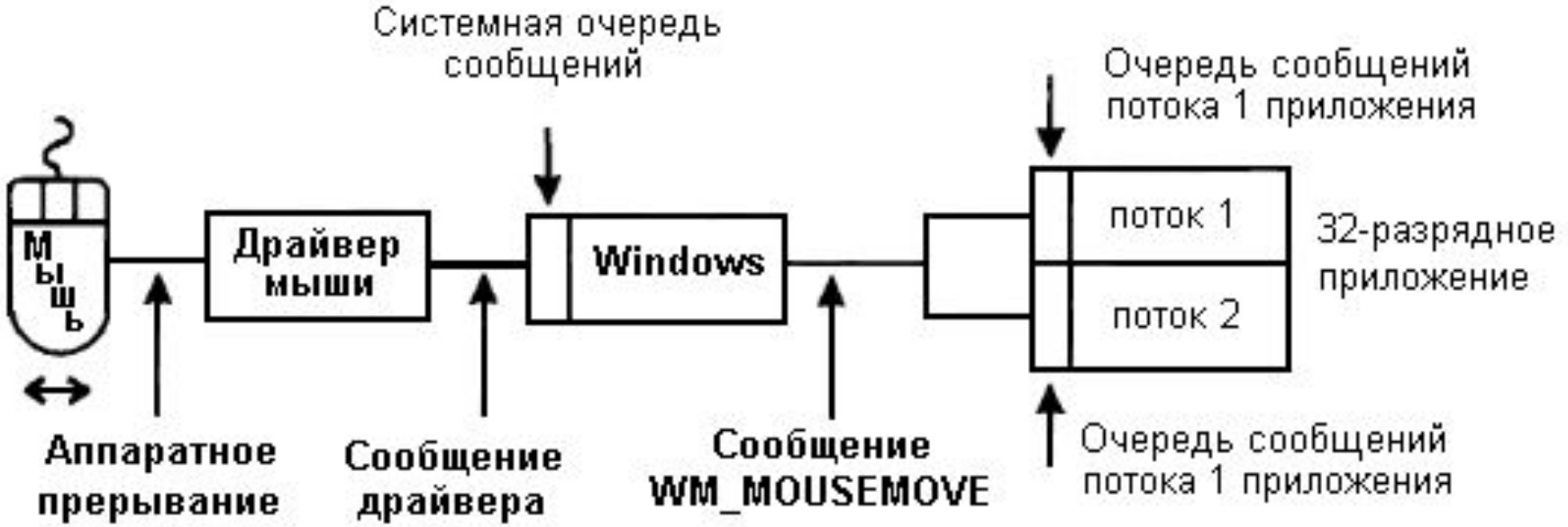
Сообщения являются реакцией системы Windows на различные происходящие в системе события: движение мыши, нажатие клавиши, срабатывание таймера и др. Отличительной особенностью сообщений является их специальный код. Для системных сообщений зарезервировано значение кода от 1 до 0x3FF.



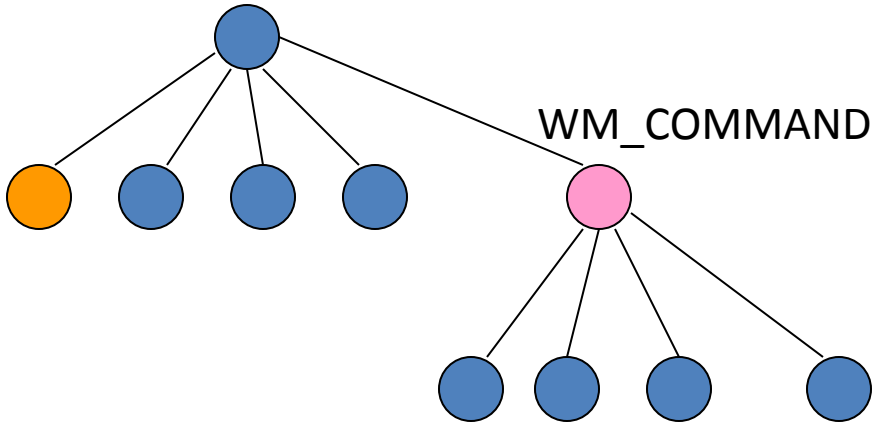
https://www.visualstudio.com/ru/vs/community/

Организация обработки сообщений

Процедура создания и пересылки сообщений от мыши:



Примеры сообщений:
WM_LBUTTONDOWN, WM_TIMER, WM_SIZE, WM_ERASEBKGND, WM_COMMAND.



Событийное управление

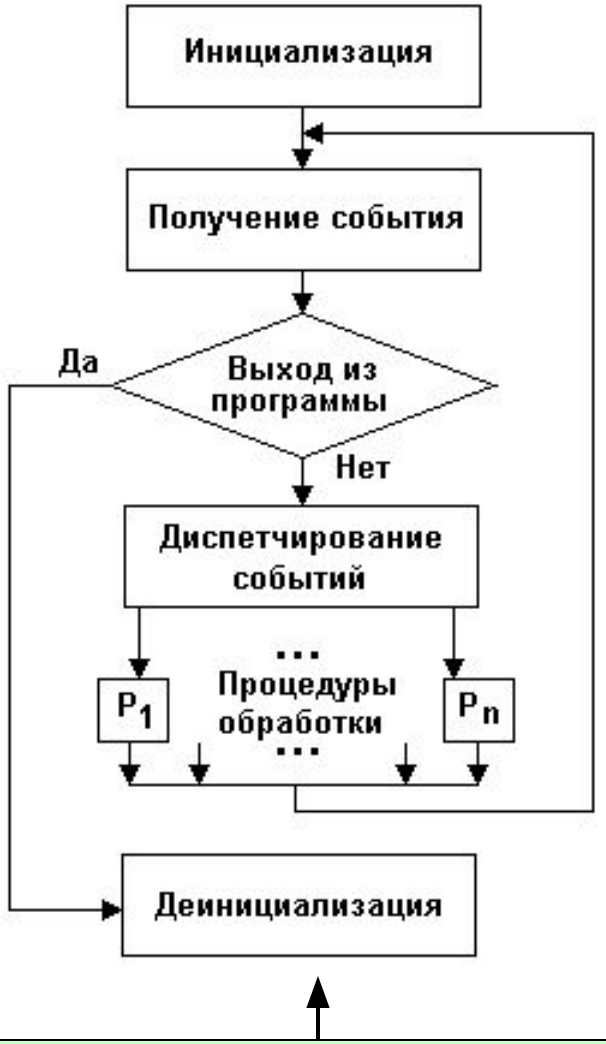
Прием сообщения – *событие*.

Характеристики события в Windows:

- Получатель (кому адресовано)
- Тип, связанный с источником
- Время возникновения
- Положение на экране курсора мыши
- Дополнительные параметры, специфичные для каждого конкретного случая

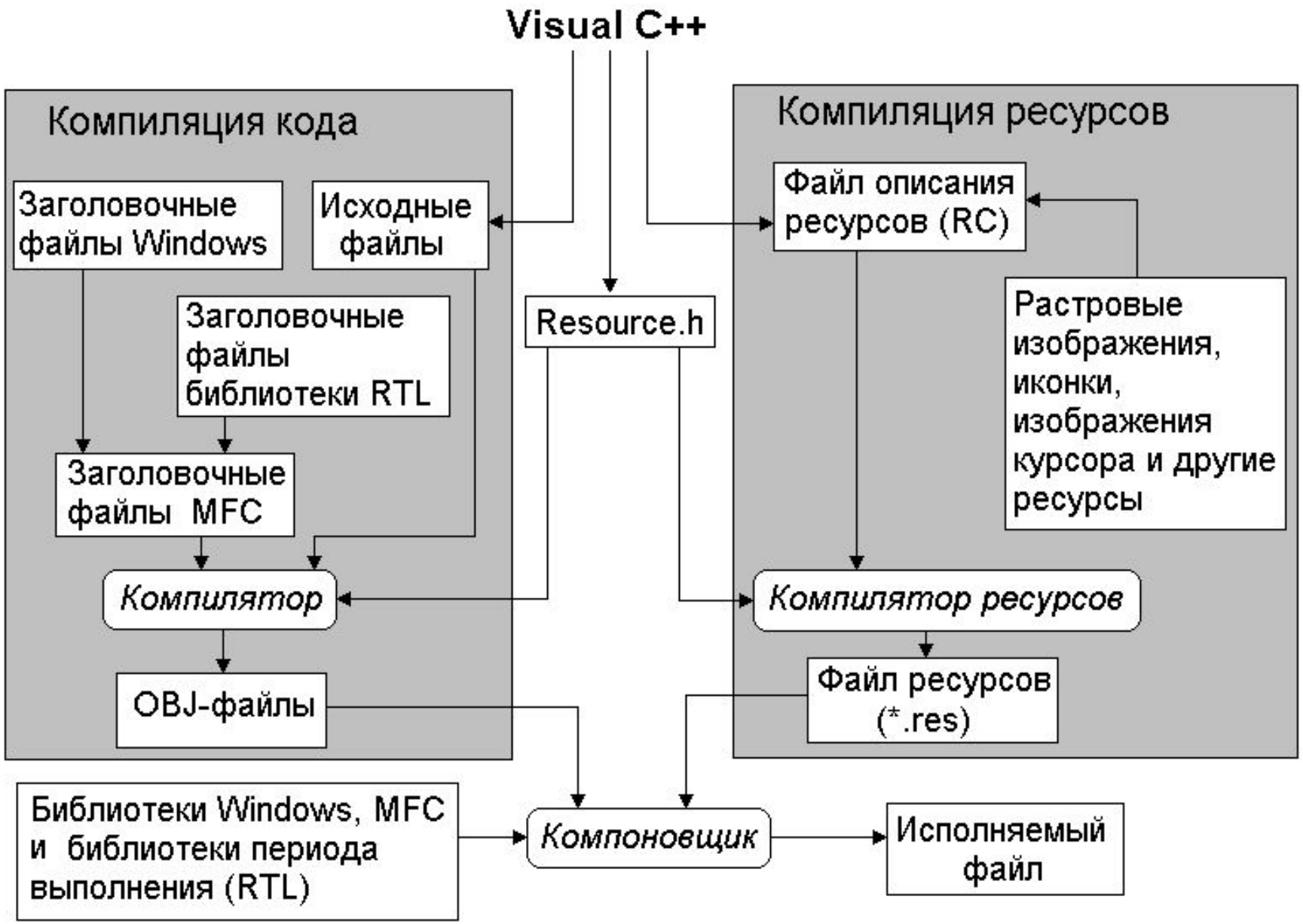
Цикл обработки сообщений организуется при помощи функций ***GetMessage*** (***PeekMessage***) и ***DispatchMessage***.

При создании приложений в среде VS – цикл организуется автоматически и не требует от программиста каких-либо действий

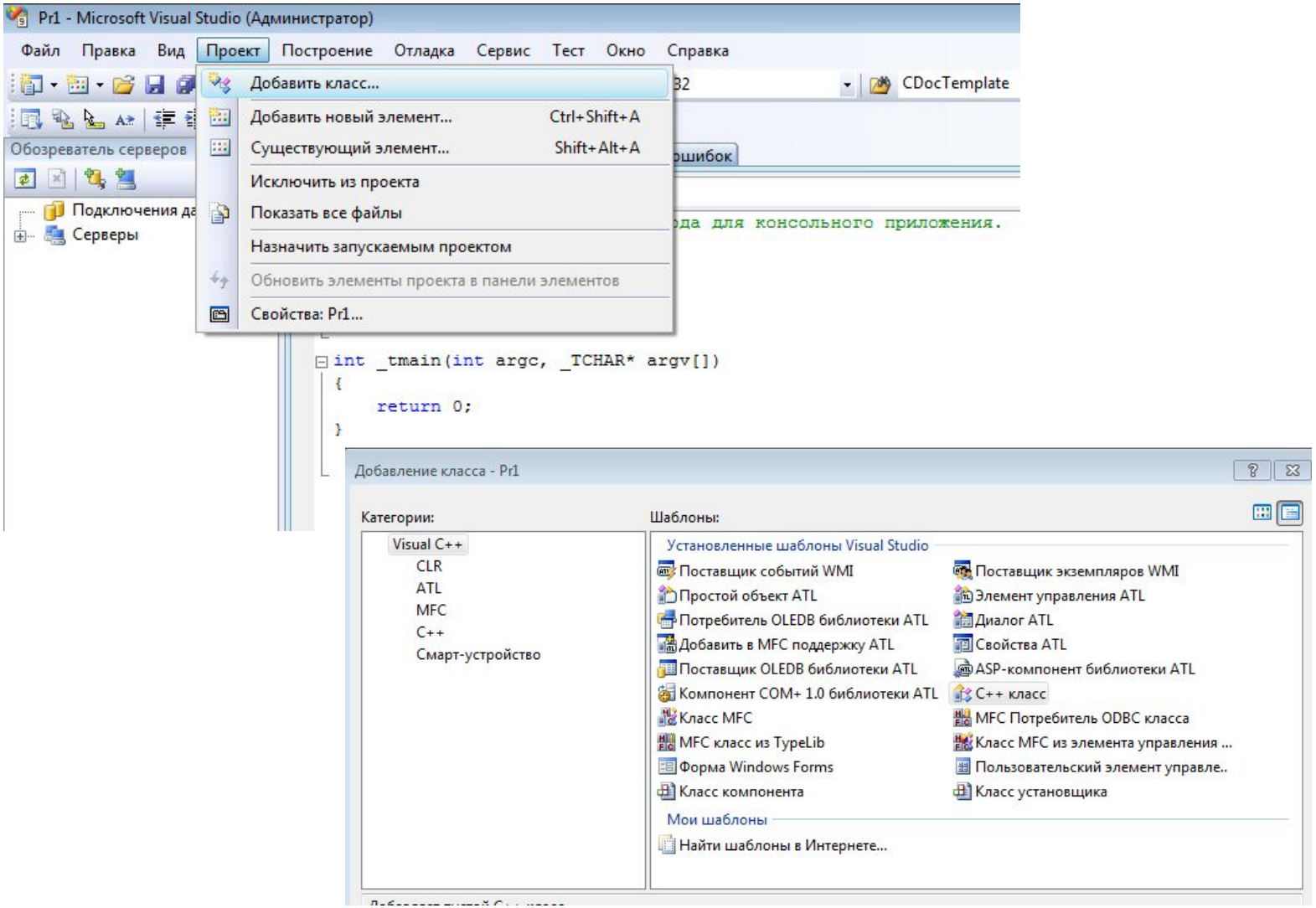


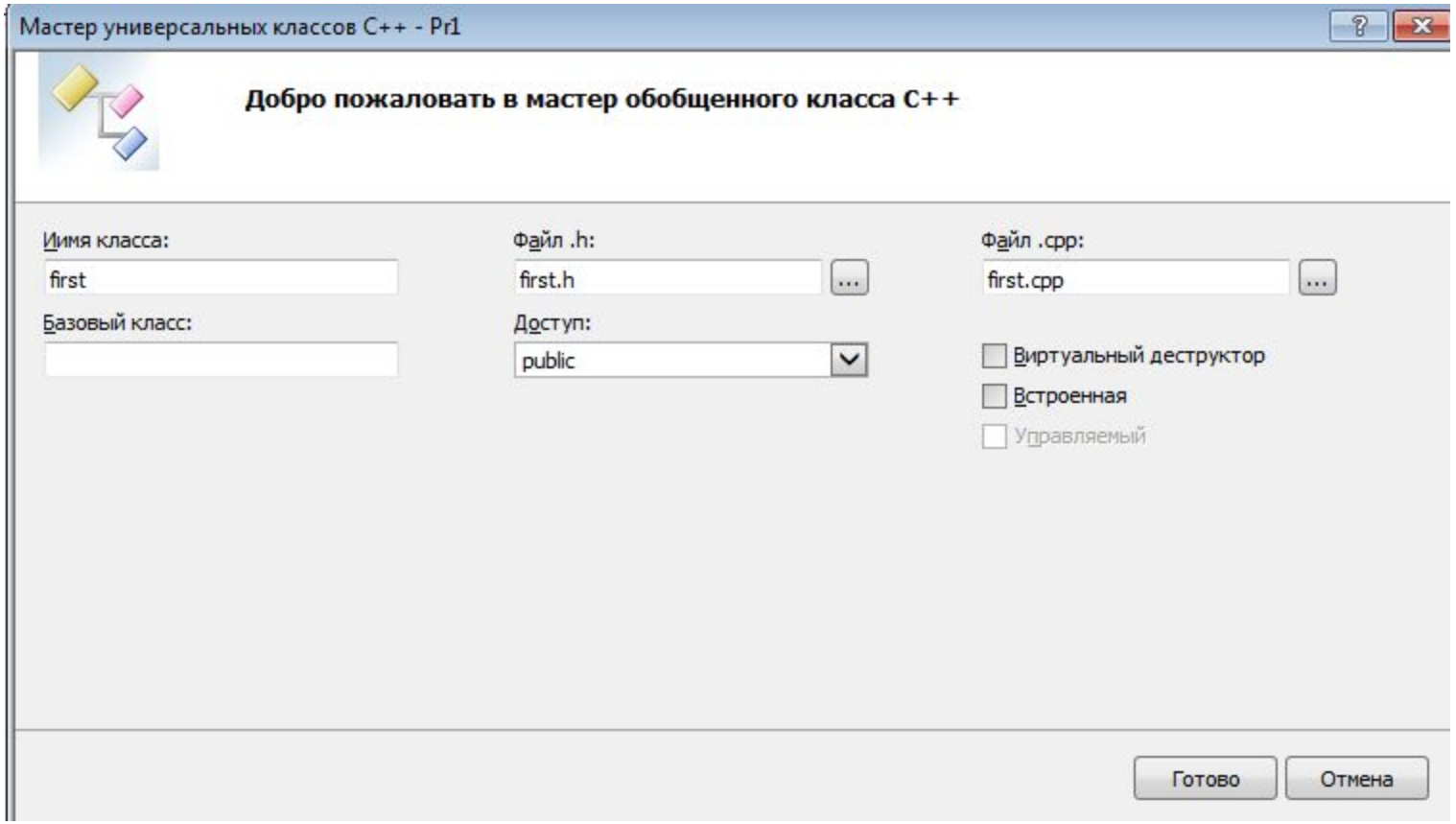
Программа с событийно-управляемой архитектурой

Процесс построения программ



Создание класса при помощи Мастера




Мастер универсальных классов C++ - Pr1
Добро пожаловать в мастер обобщенного класса C++
Имя класса:
first
Базовый класс:
Файл .h:
first.h
Доступ:
public
Файл .cpp:
first.cpp
Виртуальный деструктор
Встроенная
Управляемый
Готово
Отмена

Создание класса при помощи Мастера

Файл first.h:

```
#pragma once

class first
{
public:
  first (void);       // Конструктор
  ~first(void);     // Деструктор
};
```

Файл first.cpp:

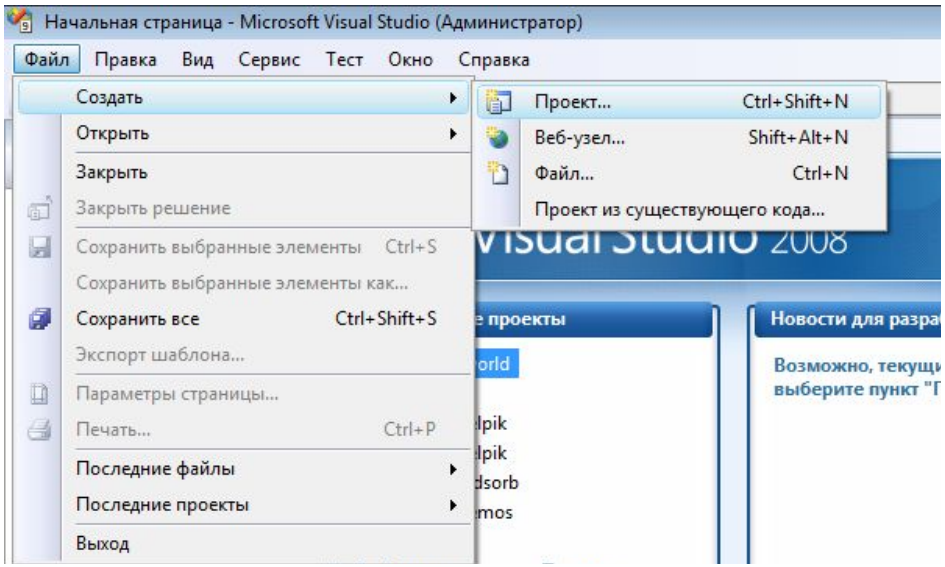
```
#include "StdAfx.h"
#include “first.h"

first::first(void)
{
}

first::~first(void)
{
}
```

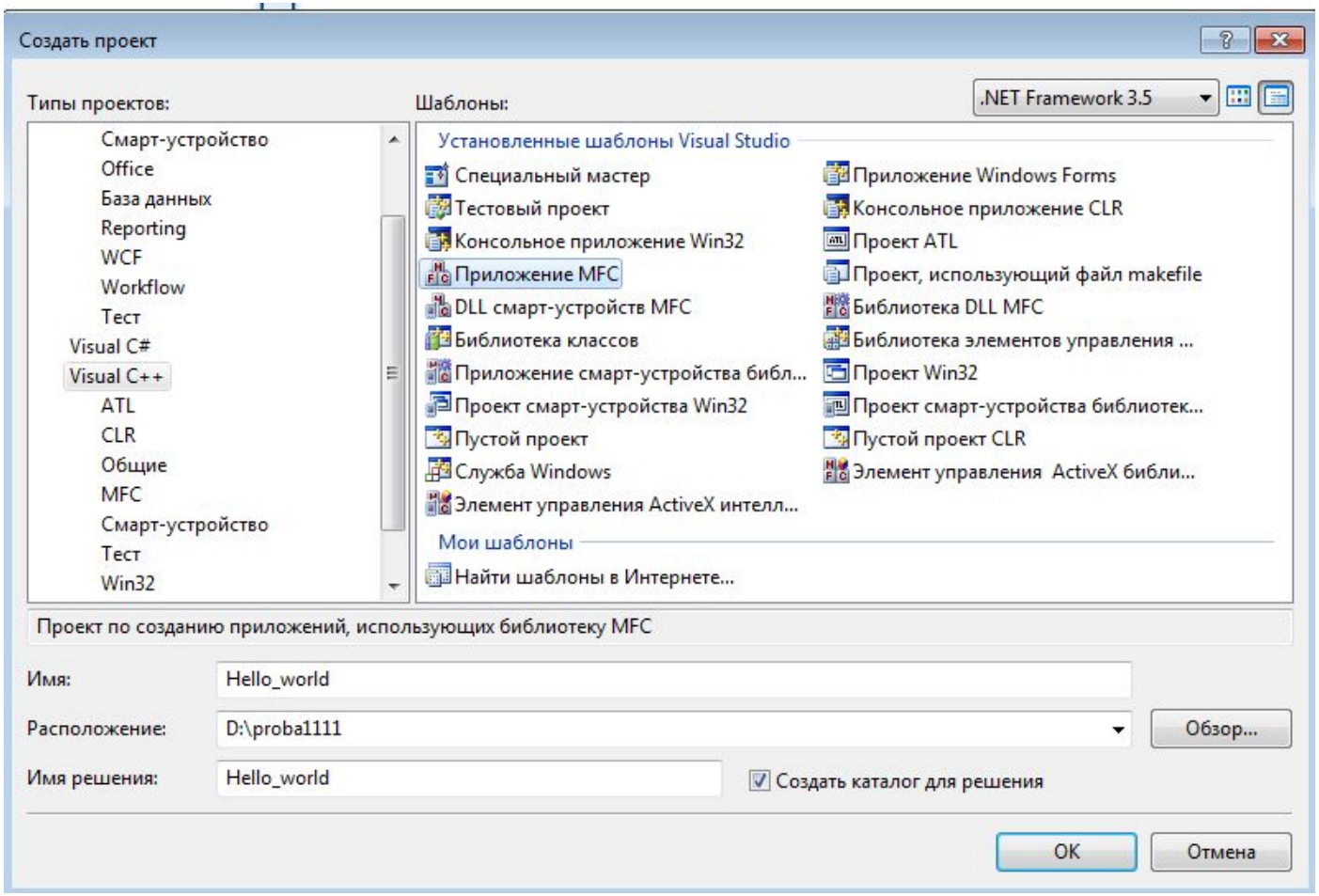
Среда разработки Microsoft Visual C++ 8.0. Создание простейшей программы средствами MFC.

1) Выберите команду верхнего меню

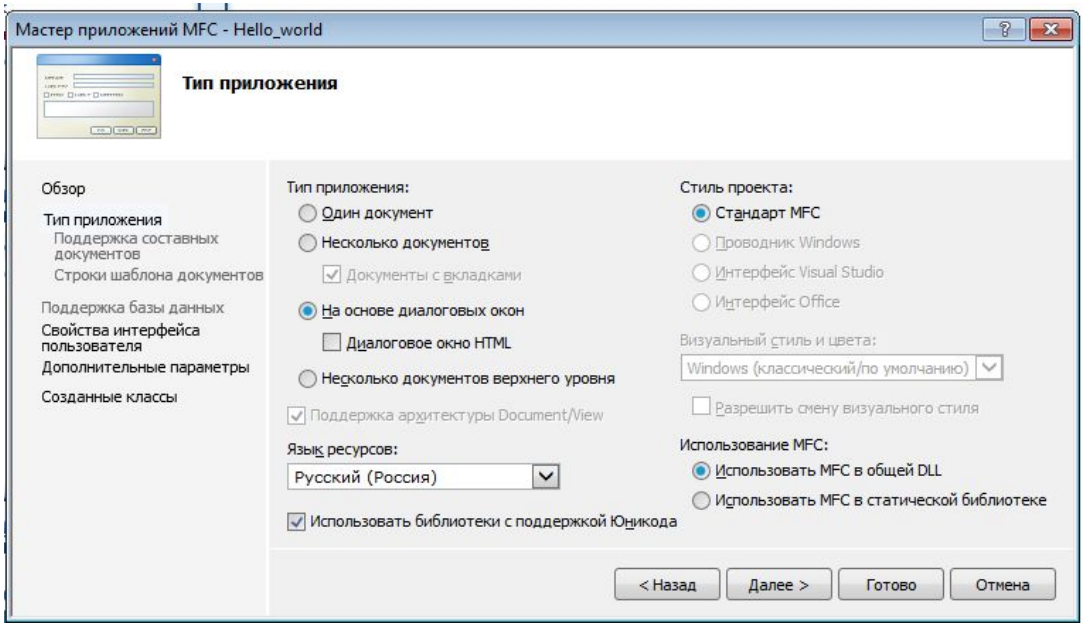


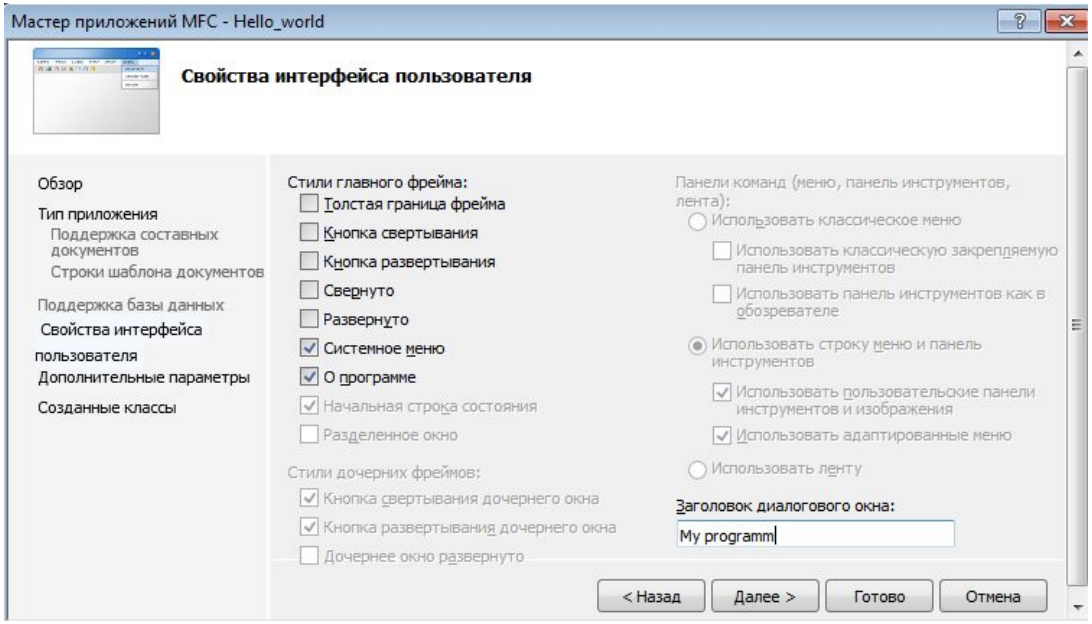
Project (vcproj) – отдельный проект, исходное состояние которого генерируется при помощи мастера Application Wizard. Проект содержит информацию о версии языка, используемой платформе, конфигурации и других особенностей, заказываемых пользователем при создании проекта

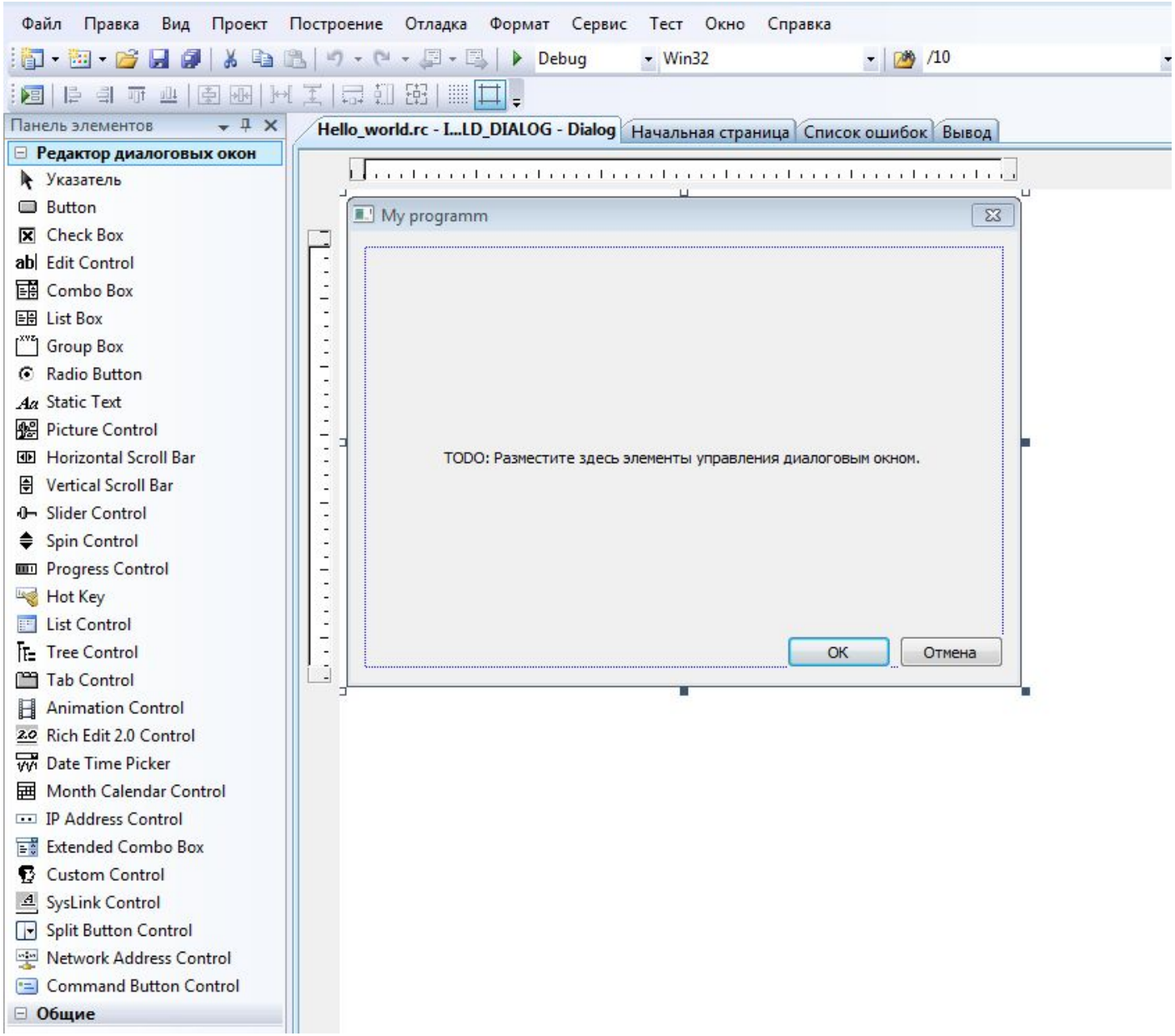
2) Выберите проект типа MFC



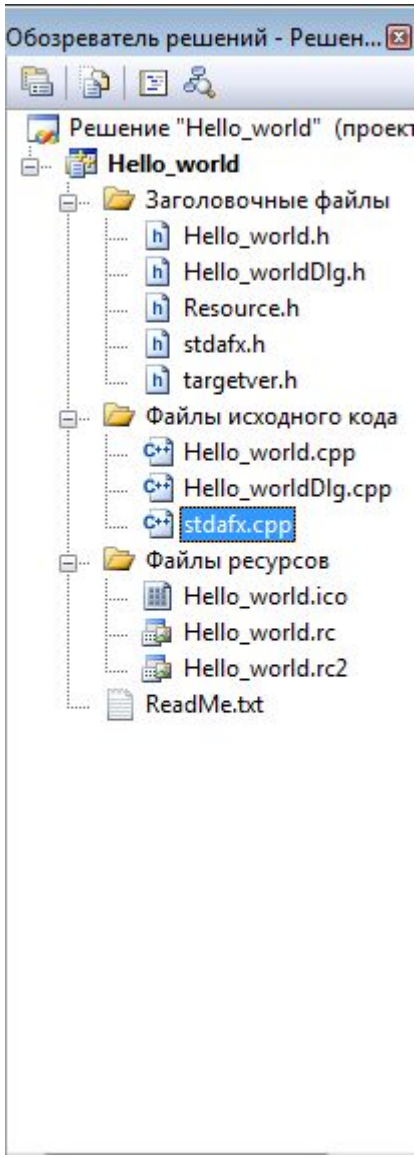
3)


Мастер приложений MFC - Hello_world
Тип приложения
Обзор
Тип приложения
Поддержка составных документов
Строки шаблона документов
Поддержка базы данных
Свойства интерфейса пользователя
Дополнительные параметры
Созданные классы
Тип приложения:
Один документ
Несколько документов
Документы с вкладками
На основе диалоговых окон
Диалоговое окно HTML
Несколько документов верхнего уровня
Поддержка архитектуры Document/View
Язык ресурсов:
Русский (Россия)
Использовать библиотеки с поддержкой Юникода
Стиль проекта:
Стандарт MFC
Проводник Windows
Интерфейс Visual Studio
Интерфейс Office
Визуальный стиль и цвета:
Windows (классический/по умолчанию)
Разрешить смену визуального стиля
Использование MFC:
Использовать MFC в общей DLL
Использовать MFC в статической библиотеке
< Назад
Далее >
Готово
Отмена


Мастер приложений MFC - Hello_world
Свойства интерфейса пользователя
Обзор
Тип приложения
Поддержка составных документов
Строки шаблона документов
Поддержка базы данных
Свойства интерфейса пользователя
Дополнительные параметры
Созданные классы
Стили главного фрейма:
Толстая граница фрейма
Кнопка свертывания
Кнопка развертывания
Свернуто
Развернуто
Системное меню
О программе
Начальная строка состояния
Разделенное окно
Стили дочерних фреймов:
Кнопка свертывания дочернего окна
Кнопка развертывания дочернего окна
Дочернее окно развернуто
Панели команд (меню, панель инструментов, лента):
Использовать классическое меню
Использовать классическую закрепляемую панель инструментов
Использовать панель инструментов как в обозревателе
Использовать строку меню и панель инструментов
Использовать пользовательские панели инструментов и изображения
Использовать адаптированные меню
Использовать ленту
Заголовок диалогового окна:
My programm
< Назад
Далее >
Готово
Отмена


Файл Правка Вид Проект Построение Отладка Формат Сервис Тест Окно Справка
Debug
Win32
/10
Панель элементов
Редактор диалоговых окон
Указатель
Button
Check Box
Edit Control
Combo Box
List Box
Group Box
Radio Button
Static Text
Picture Control
Horizontal Scroll Bar
Vertical Scroll Bar
Slider Control
Spin Control
Progress Control
Hot Key
List Control
Tree Control
Tab Control
Animation Control
Rich Edit 2.0 Control
Date Time Picker
Month Calendar Control
IP Address Control
Extended Combo Box
Custom Control
SysLink Control
Split Button Control
Network Address Control
Command Button Control
Общие
Hello_world.rc - I...LD_DIALOG - Dialog
Начальная страница
Список ошибок
Вывод
My programm
TODO: Разместите здесь элементы управления диалоговым окном.
OK
Отмена

Создание приложения MFC Application



БИБЛИОТЕКА MICROSOFT FOUNDATION CLASS: обзор проекта Hello_word

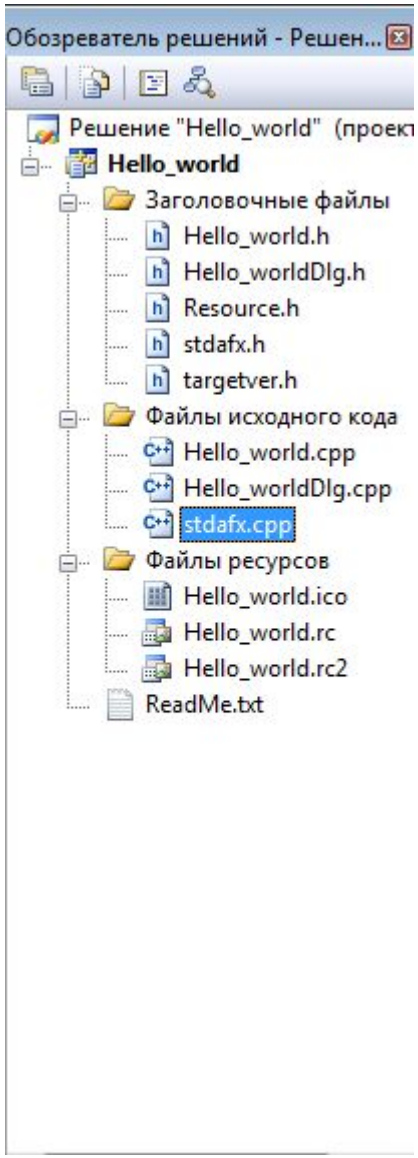
==========

Данное приложение MFC1 создано мастером приложений. Это приложение показывает основы использования фундаментальных классов Microsoft, а также является отправной точкой для создания конкретного приложения. В этом файле содержится краткое описание содержимого файлов, составляющих конкретное приложение MFC1.

Hello_word.vcproj

Это основной файл проекта для проектов VC++, создаваемых с помощью мастера приложений. В нем содержатся сведения о версии Visual C++, создавшей файл, а также сведения о платформах, конфигурациях и свойствах проекта, выбранных с помощью мастера приложений.

Создание приложения MFC Application



Hello_word.h

Это основной файл заголовка для приложения. В него включены другие определенные для проекта заголовки (в том числе Resource.h) и объявлен класс приложения CHello_worldApp.

Hello_word.cpp

Это основной исходный файл приложения, содержащий класс приложения CHello_worldApp.

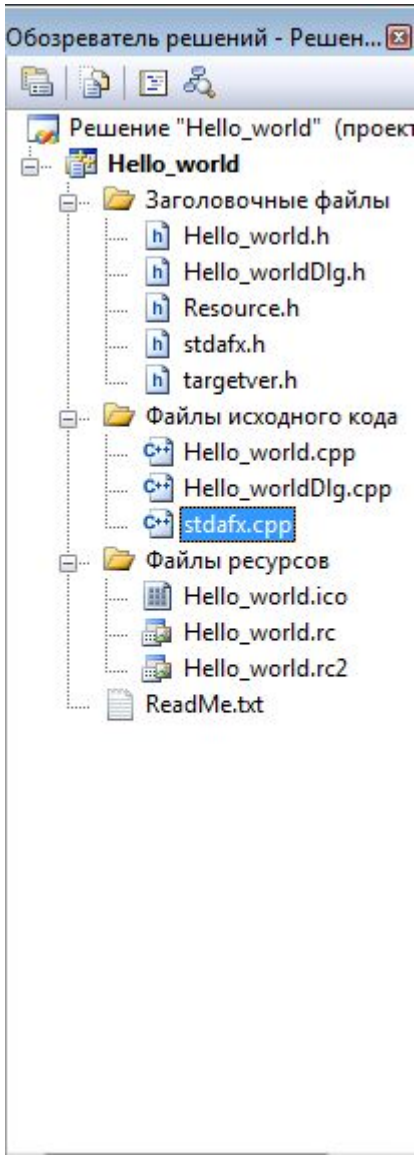
Hello_word.rc

Это список всех ресурсов Microsoft Windows, используемых программой. В него включены значки, рисунки и курсоры, хранящиеся в подкаталоге RES. Этот файл можно редактировать непосредственно в Microsoft Visual C++.

Hello_word.ico

Это файл значка, используемого в качестве значка приложения. Этот значок включен посредством основного файла ресурсов Hello_word.rc.

Создание приложения MFC Application



Hello_word.rc2

Этот файл содержит ресурсы, не редактируемые в Microsoft Visual C++. Все ресурсы, не редактируемые редактором ресурсов, следует поместить в этот файл.

Hello_word.ico

Это файл значка MFC

Resource.h

Это стандартный файл заголовка, определяющий новые идентификаторы ресурсов. Microsoft Visual C++ прочиты-вает и обновляет этот файл.

Hello_wordDlg.h - в этом файле перечислены другие включаемые файлы и описан главный класс приложения CDlgApp

Hello_wordDlg.cpp - основной файл приложения. В нем определены методы основного класса приложения CDlgApp

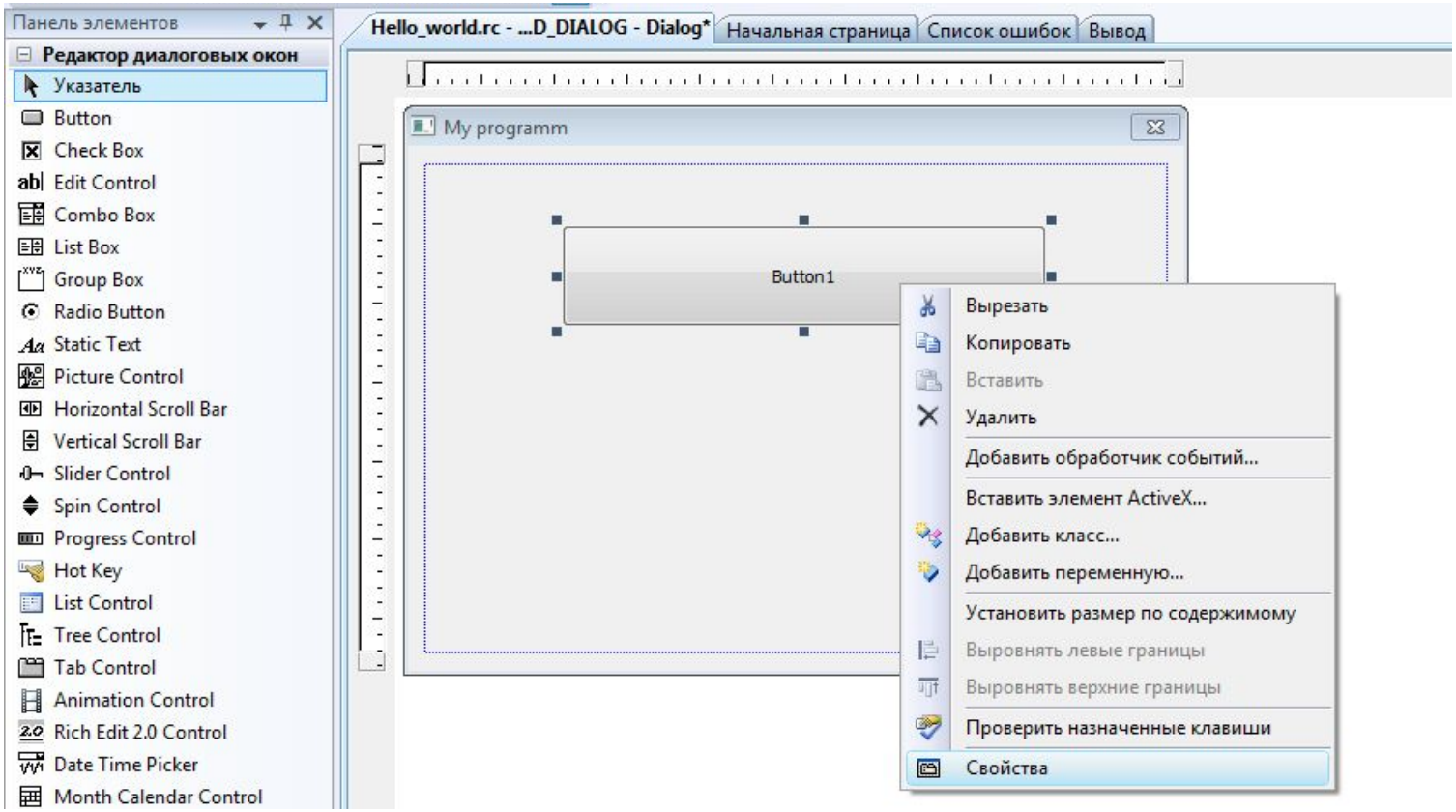
StdAfx.h, StdAfx.cpp - использование этих файлов позволяет ускорить процесс повторного построения проекта

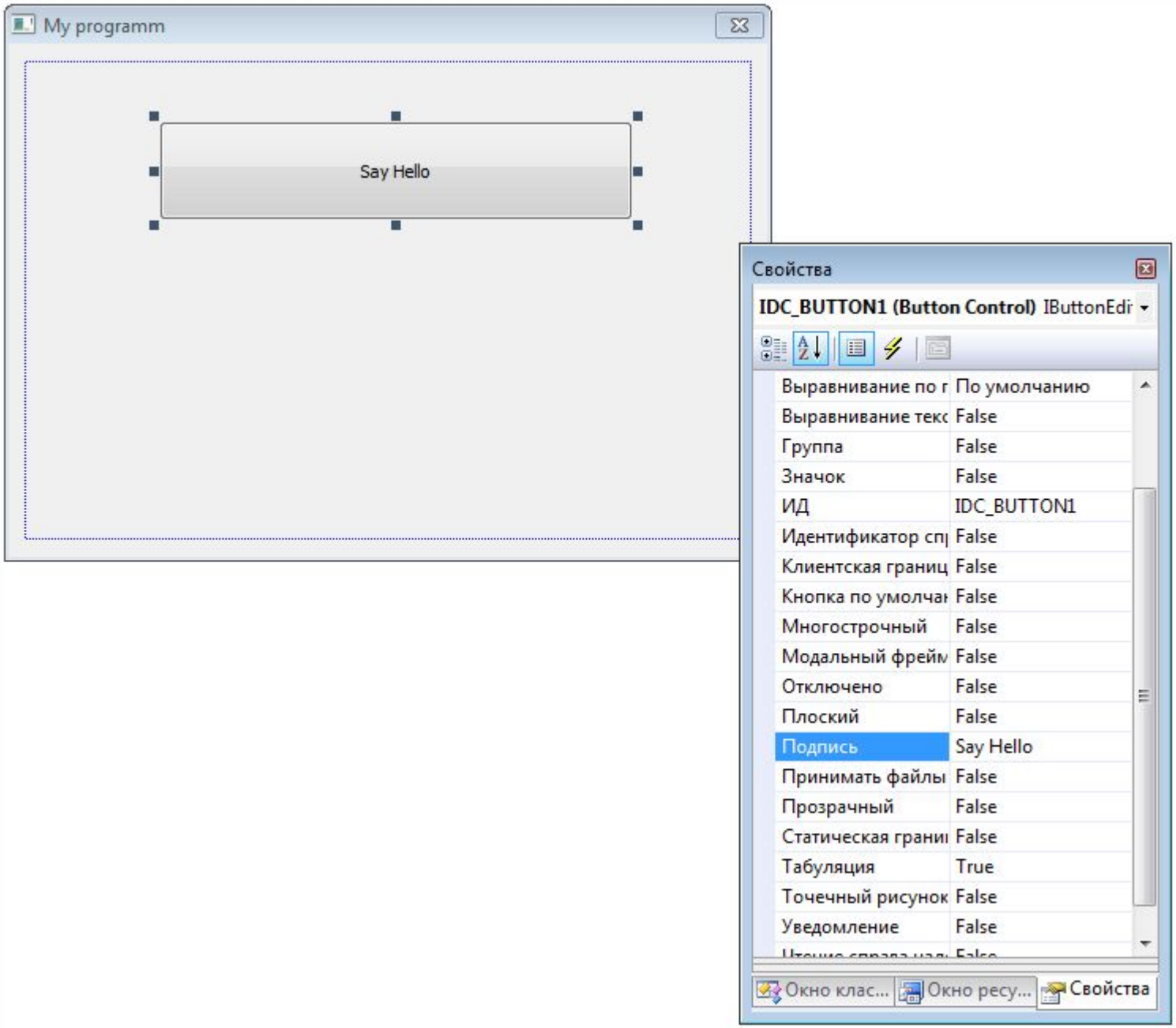
readme.txt - текстовый файл, содержащий описание проекта. В нем кратко рассмотрен каждый файл, входящий в проект, перечислены классы приложения, а также представлена другая информация.

targetver.h- позволяет использовать специфические свойства Windows

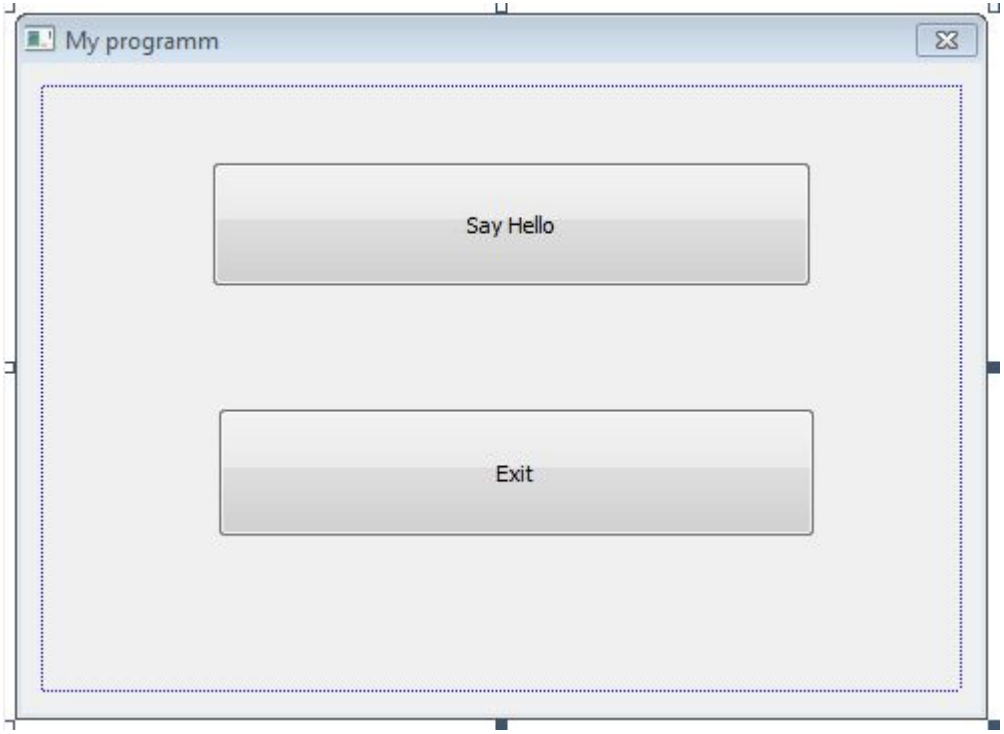
Удалите из диалоговой панели текст: "TODO:.." и две кнопки, для этого щелкните на эти элементы и нажмите DEL. Теперь у вас чистая панель и вы можете спроектировать ее на свой вкус.

Создаем кнопку с помощью панели инструментов и вызываем ее свойства по ПКМ

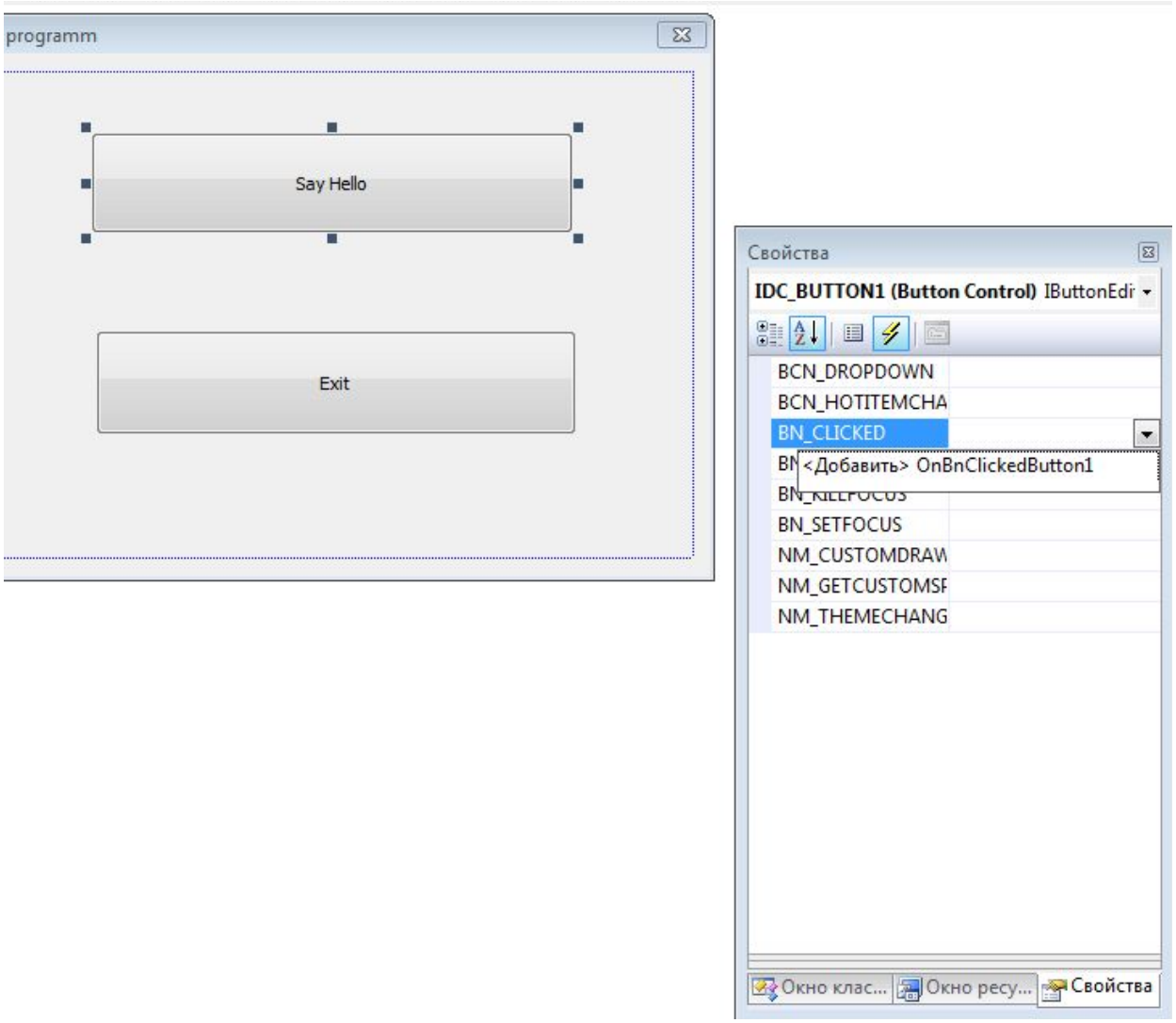


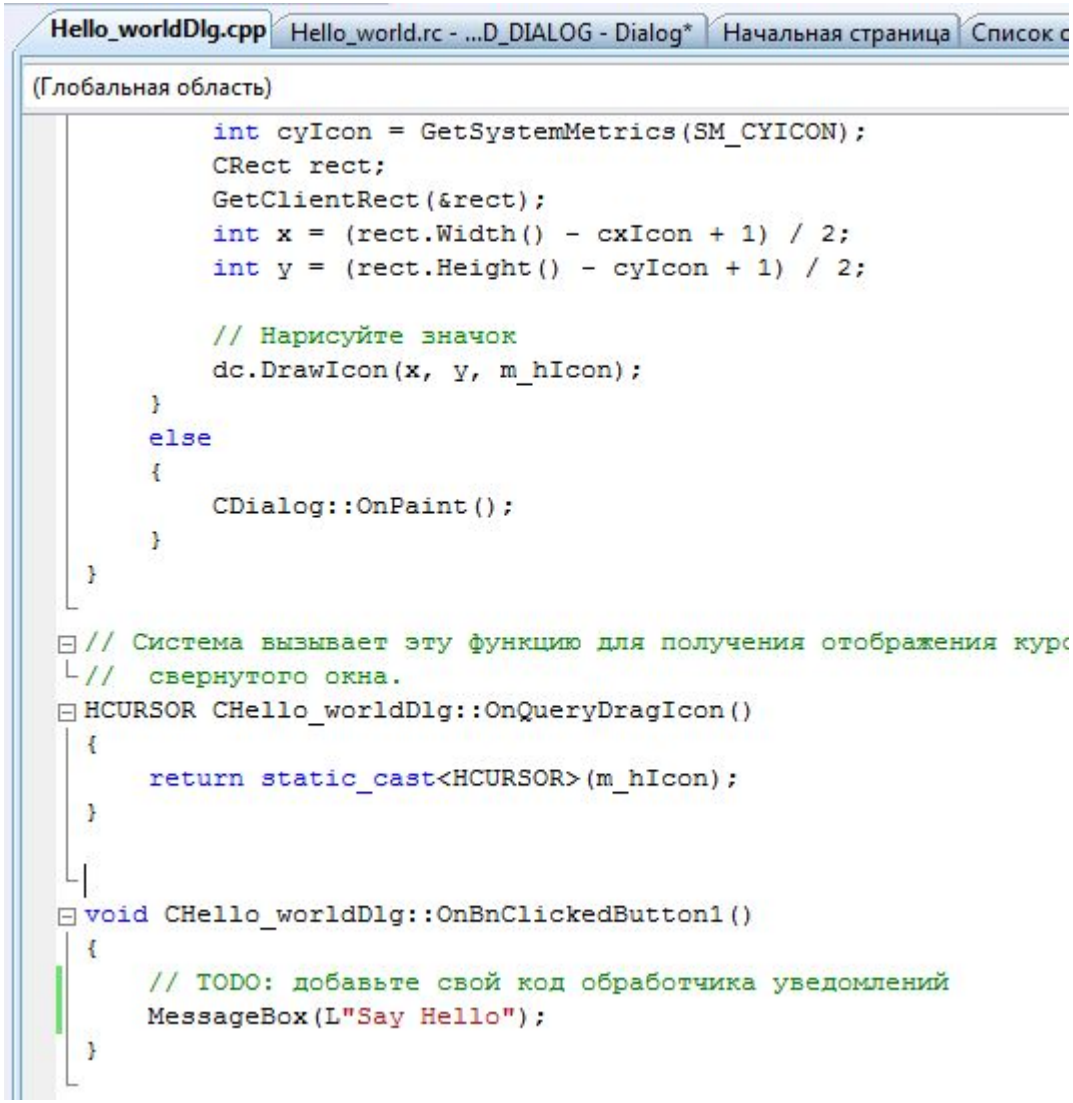

My programm
Say Hello
Свойства
IDC_BUTTON1 (Button Control) IButtonEdi
Выравнивание по г По умолчанию
Выравнивание текс False
Группа False
Значок False
ИД IDC_BUTTON1
Идентификатор сп False
Клиентская границ False
Кнопка по умолча False
Многострочный False
Модальный фрейм False
Отключено False
Плоский False
Подпись Say Hello
Принимать файлы False
Прозрачный False
Статическая грани False
Табуляция True
Точечный рисунок False
Уведомление False
Окно клас...
Окно ресу...
Свойства

Добавляем кнопку
Exit



Связывание элементов управления




Hello_worldDlg.cpp
Hello_world.rc - ...D_DIALOG - Dialog*
Начальная страница
Список с
(Глобальная область)
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;
GetClientRect(&rect);
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;
// Нарисуйте значок
dc.DrawIcon(x, y, m_hIcon);
}
else
{
CDialog::OnPaint();
}
}
// Система вызывает эту функцию для получения отображения курс
// свернутого окна.
HCURSOR CHello_worldDlg::OnQueryDragIcon()
{
return static_cast<HCURSOR>(m_hIcon);
}
void CHello_worldDlg::OnBnClickedButton1()
{
// TODO: добавьте свой код обработчика уведомлений
MessageBox(L"Say Hello");
}

Функция MessageBox("Say Hello"); вызывает окно сообщений, в котором написан текст Say Hello, который и является параметром этой функции. Функция MessageBox() может принимать три параметра. Первый - это само сообщение, второй – заголовок сообщения, третий – комбинация флагов (см. табл. 1.1,1.2,1.3).

Таблица 1.1. Комбинация кнопок MESSAGEBOX()

ID	Кнопки
MB_ABORTRETRYIGNORE	Abort, Retry, Ignore
MB_OK	OK
MB_OKCANCEL	OK, Cancel
MB_RETRYCANCEL	Retry, Cancel
MB_YESNO Yes, No	Yes, No
MB_YESNOCANCEL	Yes, No, Cancel

Таблица 1.2. Иконки MESSAGEBOX().

ID	Иконки
MB_ICONINFORMATION	Informational icon
MB_ICONQUESTION	Question mark icon
MB_ICONSTOP	Stop sign icon
MB_ICONEXCLAMATION	Exclamation mark icon

Таблица 1.3. Значения, возвращаемые MESSAGEBOX().

ID	Ответ после нажатия
IDABORT	Abort
IDRETRY	Retry
IDIGNORE	Ignore
IDYES	Yes
IDNO	No
IDOK	OK
IDCANCEL	Cancel

Затем связываем кнопку

Exit

```
void CHello_worldDlg::OnBnClickedButton2()
{
    // TODO: добавьте свой код обработчика уведомлений
    OnOK();
}
```

Наиболее существенным обстоятельством в развитии методологии ООП явилось осознание того, что процесс написания программного кода может быть отделен от процесса проектирования структуры программы.

До написания кода необходимо провести

- общий анализ требований к будущей программе
- анализ конкретной предметной области, для которой разрабатывается программа.

Все это привело к появлению специальной методологии - методологии **объектно-ориентированного анализа и проектирования (ООАП).**