# ПМО ИУС

# Обработка ошибочных ситуаций. Исключения.

Лекция 11

# Понятие ошибочной ситуации

```
float
div( float a, float b )
{
   // Разработчик функции.

   if ( b == 0 ) {
       // Поведение в случае ошибочной ситуации. ??
   }

   return a / b;

}

int main()
{
[…]
   // Пользователь функции.
   float c = div( 1, 2 );
[…]
}
```

# Прекращение выполнения программы

```
float
div( float a, float b )
{
   // Разработчик функции.

   if ( b == 0 ) {
         // Прекращение с кодом ошибки 1.
         exit(1);
   }

   return a / b;
}

int main()
{
[…]
   // Пользователь функции.
   float c = div( 1, 2 );
[…]
}
```

# Возвращение значения кода ошибки

```
/*!
   Проводит соединение с БД.
   \return корректность выполнения функции.
   \retval 0 все в порядке.
   \retval 1 соединение уже было установлено.
   \retval 2 ошибка связи.
*/
int
connect_to_db( const std::string & db_name )
{
[…]
}

int main()
{
[…]
   int res = connect_to_db(“test_db”);
[…]
}
```

# Возвратить допустимое значение и оставить программу в ненормальном состоянии

```
float
div( float a, float b )
{
   // Разработчик функции.

   if ( b == 0 ) {
         // Ошибка – игнорируем. Считаем деление на 1.
         return a;
   }

   return a / b;
}

int main()
{
[…]
   // Пользователь функции.
   float c = div( 1, 2 );
[…]
}
```

# Вызвать функцию, предназначенную для обработки ошибочных ситуаций

```
void
assert_true( bool value, const std::string & msg )
{
    // False - пропускаем.
    if ( !value )
          return;

    std::cerr << msg << std::endl;
    // Дополнительные специфичные действия.
}

float
div( float a, float b )
{
    // Разработчик функции.

    assert_true( b == 0, “Divide by zero” );

    return a / b;
}

int main()
{
[…]
    // Пользователь функции.
    float c = div( 1, 2 );
[…]
}
```

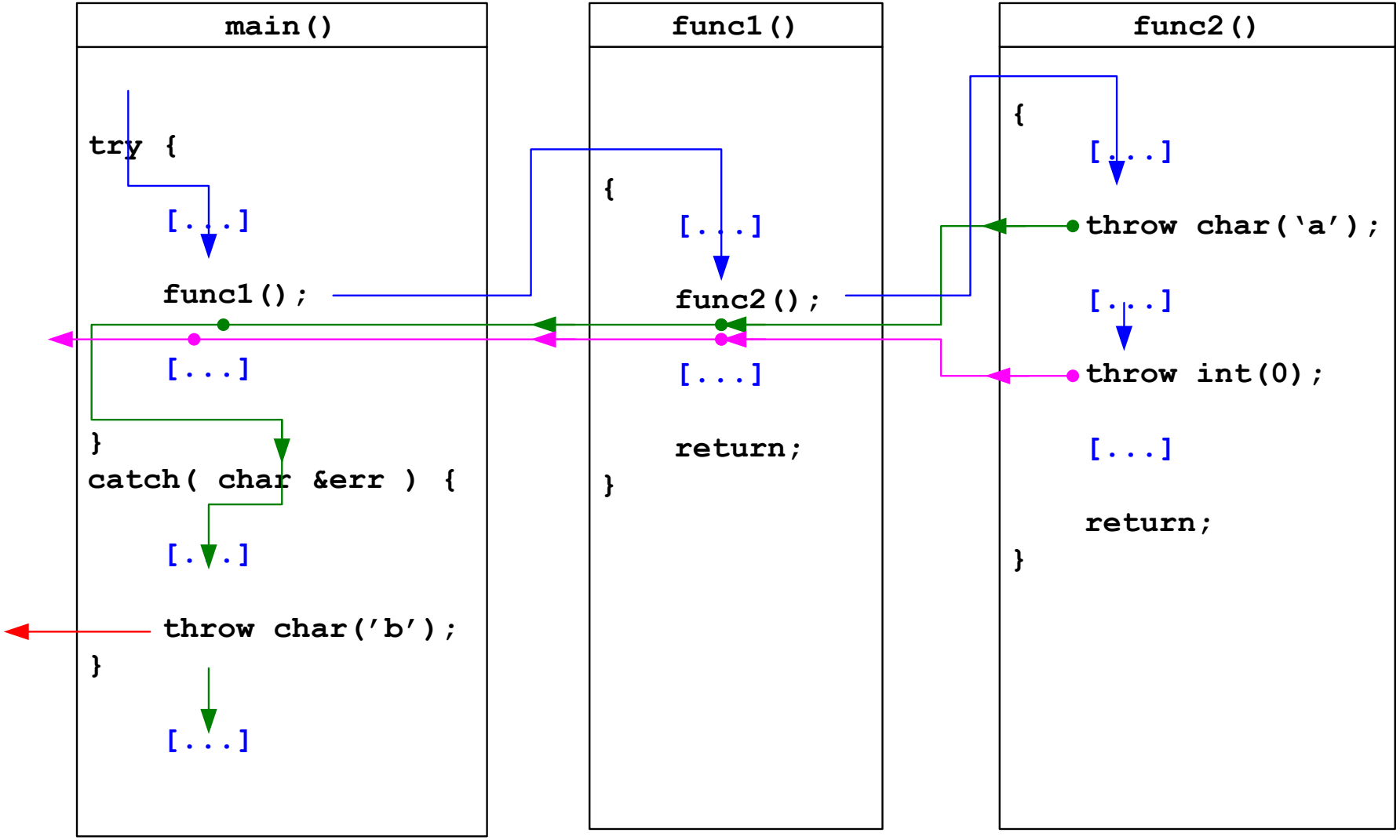
# Исключения

```
float
div( float a, float b )
{
    if ( b == 0 ) {
          // Функция, обнаружившая проблему, генерирует исключение в надежде, что
          // вызвавшая её функция сможет решить возникшую проблему.
          throw int(0);
    }

    return a / b;
}

int main()
{
[…]
    // Функция, которая может решать проблемы данного типа,
    // перехватывает (catch) такие исключения.
    try {
          float c = div( x, y );
    }
    catch( const int & ex ) {
        // Обработка ошибок типа int.
        if ( ex == 0 ) {
            std::cerr << "Divide by zero." << std::endl;
        }
        else {
            // Обработка всех ошибок.
            std::cerr << "Error " << err << std::endl;
        }
    }
}
```

# Синтаксис исключений

```
try
{
   // Блок, где может возникнуть ошибка
   // (исключение) – блок повторных попыток.
   // Содержит код, в котором предполагается
   // возможное появление ошибочных ситуаций.
}
catch( <тип исключения> )
{
   // Блок обработки ошибок данного типа
   // (обработчик исключительных ситуаций,
   // улавливающий блок).
}

// Генерация исключений.
throw <объект исключения>;
```

# Обработка исключительной ситуации



# Перехват исключений catch

```
try
{
[…]
}
catch( const int & ex )
{
   // Обработка исключений типа int.
[…]
}
catch( … )
{
   // Обработка всех исключений, кроме типа int.
[…]
}
```

# Использование классов исключений

```
class math_ex_t;
class overflow_ex_t: public math_ex_t;
class divide_by_zero_ex_t: public math_ex_t;
class underflow_ex_t: public math_ex_t;

try
{
    […]
}
catch( const divide_by_zero_ex_t & ex )
{
    std::cerr << “Divide by zero” << std::endl;
}
catch( const math_ex_t & ex )
{
    std::cerr << “Math error” << std::endl;
}
catch( … )
{
    std::cerr << “Unknown error” << std::endl;
}
```

# Повторная генерация исключений

```
try
{
   […]
}
catch( const math_ex_t & ex )
{
   if ( .. )
   {
        // Можем обработать ошибку.
        […]
   }
   else
   {
        // Не можем. Пробрасывается объект ex дальше.
        throw;
   }
}
```

# Исключения std::exception

```
try
{
[…]
  throw std::runtime_error("General Failure");
[…]
}
catch( const std::exception & ex )
{
   std::cout <<
      "std::exception : " <<
      ex.what() <<
      std::endl;
}
```

# Исключения C++Builder

```
// EConvertError – ошибка преобразования типов.

// EMathError:
//   EOverflow
//   EUnderflow
//   EZeroDivide

try
{
   int x = StrToInt( Edit1->Text );
[…]
}
catch( const ConvertError & p )
{
   ShowMessage( "Ошибка конвертации, описание:" +
       p.Message() );
}
```

# Создание собственных классов исключений

```
//! Класс пользовательских исключений.
class example_ex_t
{
   public:
      example_ex_t( const std::string & msg )
      : m_msg(msg) { }

      //! Доступ к данным объекта исключения.
      //! return что произошло.

      std::string
      msg()
      {
          return m_msg;
      }
   private:
      //! Внутреннее хранимое поле данных.
      std::string m_msg;
};
```

```
try
{

    […]

    throw example_ex_t("Overflow!");

    […]

}
catch( const example_ex_t & ex )
{
    std::cerr <<
        ex.msg() <<
        std::endl;
}
```