Основы построения файловых систем

Разное о C++

```
std::vector<uint8_t> int_to_utf8(const uint32_t symbol) {
        std::vector<uint8_t> utf8_char;

        if (symbol < 1 << PAYLOAD_BITS_IN_ONE_BYTE_UTF8_CHAR) {
                ....
        } else {
                assert(0);
        }
        ....
}
```

Разное о C++

```
std::vector<uint8_t> int_to_utf8(const uint32_t symbol) {
        std::vector<uint8_t> utf8_char;

        if (symbol < 1 << PAYLOAD_BITS_IN_ONE_BYTE_UTF8_CHAR) {
                ....
        } else {
                assert(0);
        }
        ....
}
```

```
std::vector<uint8_t> int_to_utf8(const uint32_t symbol) {
        std::vector<uint8_t> utf8_char;

        if (symbol < 1 << PAYLOAD_BITS_IN_ONE_BYTE_UTF8_CHAR) {
                ....
        } else {
                throw std::runtime_error("bad input format");
        }
        ....
}
```

Разное о C++

```
std::vector<uint8_t> int_to_utf8(const uint32_t symbol) {
        std::vector<uint8_t> utf8_char;

        if (symbol < 1 << PAYLOAD_BITS_IN_ONE_BYTE_UTF8_CHAR) {
                ....
        } else {
                assert(0);
        }
        ....
}
```

```
optional<vector<uint8_t> int_to_utf8(const uint32_t symbol) {
        std::vector<uint8_t> utf8_char;

        if (symbol < 1 << PAYLOAD_BITS_IN_ONE_BYTE_UTF8_CHAR) {
                ....
        } else {
                return optional<vector<uint8_t>>();
        }
        ....
}
```

Разное о C++

```
class StrUtf8Iter {
  public:
    StrUtf8Iter(const std::vector<uint8_t>& str_utf8) :
      __str_utf8(str_utf8)
    {}

  uint32_t next_int() {
    if (this->__str_utf8.size() < this->__current_postion) {
      throw "StopIteration";
    }
  ....

while (1) {
  try {
    str_ints.push_back(str_utf8_iterator.next_int());
  }
  catch (const char* msg) {
    if (!strcmp(msg, "StopIteration")) {
      return str_ints;
    } else {
      assert (0);
    }
  }
}
```

Разное о C++

```
class StrUtf8Iter {
  public:
    StrUtf8Iter(const std::vector<uint8_t>& str_utf8) :
      __str_utf8(str_utf8)
    {}

  uint32_t next_int() {
    if (this->__str_utf8.size() < this->__current_postion) {
      throw "StopIteration";
    }
  ....

while (1) {
  try {
    str_ints.push_back(str_utf8_iterator.next_int());
  }
  catch (const char* msg) {
    if (!strcmp(msg, "StopIteration")) {
      return str_ints;
    } else {
      assert (0);
    }
  }
}
```

- Исключения не надо использовать как замену return.
- Исключения заставляют компилятор пессимизировать код.
- Исключения, не унаследованные от std::exception, будут сюрпризом для всех. Например, throw std::exception внутри atomic_cancel-блока приводит к отмене транзакции, а throw чего угодно ещё – к вызову std::abort (см. Transaction Memory TS)

Разное о C++

```
class StrUtf8Iter {
  public:
    StrUtf8Iter(const std::vector<uint8_t>& str_utf8) :
      __str_utf8(str_utf8)
    {}

  uint32_t next_int() {
    if (this->__str_utf8.size() < this->__current_postion) {
      throw "StopIteration";
    }
  ....
}

while (1) {
  try {
    str_ints.push_back(str_utf8_iterator.next_int());
  }
  catch (const char* msg) {
    if (!strcmp(msg, "StopIteration")) {
      return str_ints;
    } else {
      assert (0);
    }
  }
}
```

```
class StrUtf8Iter {
public:
  bool hasNext() const {return cur_ != str_utf8_.cend();}

  uint32_t getNext() {
    ....
  }

private:
  const vector<uin8_t>& str_utf8_;
  vector<uint8_t>::const_iterator cur_;
};

....

while (it.hasNext())
  str_ints.push_back(it.getNext());
```

Разное о C++

```
class Formatter
{
public:
    Formatter() {}
    ~Formatter() {}

    ....

private:
    std::stringstream stream_;

    Formatter(const Formatter &);
    Formatter & operator = (Formatter &);
};
```

Разное о C++

```
class Formatter
{
public:
    Formatter() {}
    ~Formatter() {}

    ....

private:
    std::stringstream stream_;

    Formatter(const Formatter &);
    Formatter & operator = (Formatter &);
};
```

```
class Formatter
{
public:
    Formatter() {}
    Formatter(const Formatter &) = delete;
    ~Formatter() {}

    ....

    Formatter & operator = (Formatter &) = delete;

private:
    std::stringstream stream_;
};
```

Разное о C++

```
#define IS_SECOND_BYTE(x) (((x >> 7) & 1) && ((x >> 6) & 1) &&\
                           (((x >> 5) & 1) == 0))
```

- x будет вычислен много раз
- Подстановки в макросах выполняются текстуально. Чему равно IS_SECOND_BYTE(0b11000000 | 0b00011111)?

Разное о C++

```
#define IS_SECOND_BYTE(x) (((x >> 7) & 1) && ((x >> 6) & 1) &&\
                           (((x >> 5) & 1) == 0))
```

```
static inline bool is_second_byte(uint8_t x)
{
  return (x & 0b11100000) == 0b11000000;
}
```