Curiously recurring template pattern

class Derived : **public** Base<Derived>

Adam Mizerski
adam@mizerski.pl

Warsaw C++ Users Group

13 stycznia 2015

The Matasano Crypto Challenges

The Matasano Crypto Challenges
http://cryptopals.com/

Convert hex to base64

The string:

49276d206b696c6c696e6720796f757220627261696e206c
696b652061207 06f69736f6e6f7573206d757368726f6f6d

Should produce:

SSdtIGtpbGxpbmcgeW91ciBicmFpbiBs
aWtlIGEgcG9pc29ub3VzIG11c2hyb29t

[0] Źródło: `http://cryptopals.com/sets/1/challenges/1/`

Czym jest base64?

Czym jest base64?

Metoda kodowania danych binarnych w ASCII

Czym jest base64?

- Ciąg znaków z zakresu `[A-Za-z0-9+/]`,
- 3 bajty to 4 znaki.

Tabela base64

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

[0] Źródło: https://en.wikipedia.org/wiki/Base64

Kodowanie base64

Text content	M								a								n							
ASCII	77 (0x4d)								97 (0x61)								110 (0x6e)							
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Index	19								22								5							
Base64-encoded	T								W								F							

[0] Źródło: https://en.wikipedia.org/wiki/Base64

Przykład

Man is distinguished, not only by his reason, but by this singular passion from other animals, which is a lust of the mind, that by a perseverance of delight in the continued and indefatigable generation of knowledge, exceeds the shortvehemence of any carnal pleasure.

```
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlz
IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg
dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu
dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo
ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=
```

[0]Źródło: https://en.wikipedia.org/wiki/Base64

Dopełnianie (padding)

opcjonalne

any carnal pleasure.		YW55IGNhcm5hbCBwbGVhc3VyZS4=
any carnal pleasure		YW55IGNhcm5hbCBwbGVhc3VyZQ==
any carnal pleasur	⇔	YW55IGNhcm5hbCBwbGVhc3Vy
any carnal pleasu		YW55IGNhcm5hbCBwbGVhc3U=
any carnal pleas		YW55IGNhcm5hbCBwbGVhcw==

[0]Źródło: https://en.wikipedia.org/wiki/Base64

Czego wymagamy od klasy Base64String?

- Tworzenie i przypisywanie z **const char**$*$ i std::string,
  - Rzuca wyjątek, gdy dane wejściowe niepoprawne,
- Iteratory i `operator[]` tylko do odczytu.

Zarys implementacji

- Klasa Base64Char
  - odpowiada za jeden znak,

Zarys implementacji

- Klasa Base64Char
  - odpowiada za jeden znak,
- Klasa Base64String
  - zawiera obiekty klasy Base64Char,
  - odpowiada za cały ciąg.

Base64Char

Część trywialna.

```
1  class Base64Char {
2      char char_;
3  public:
4      Base64Char() = default;
5      Base64Char(const Base64Char&) = default;
6      Base64Char(Base64Char&&) = default;
7      Base64Char& operator=(const Base64Char&) = default;
8      Base64Char& operator=(Base64Char&&) = default;
9      operator char() const { return char_; }
```

Base64Char

Rzuca wyjątek, gdy dane wejściowe niepoprawne.

```
10  private:
11      friend class Base64String;
12      static void throw_if_illegal(char char_);
13  public:
14      Base64Char(char char__) : char_(char__)
15      { throw_if_illegal(char__); }
16
17      Base64Char& operator=(char char__)
18      {
19          throw_if_illegal(char__); // exception safety
20          char_ = char__;
21          return *this;
22      }
```

Base64Char

Tylna furtka.

```
23  private:
24      friend class Base64String; // dla przypomnienia
25      static Base64Char create_unchecked(char char__) {
26          auto c = Base64Char{};
27          c.char_ = char__;
28          return c;
29      }
30  };
```

Base64String

Część trywialna.

```
1  class Base64String {
2  private:
3      std::basic_string<Base64Char> value;
4  public:
5      Base64String() = default;
6      Base64String(const Base64String&) = default;
7      Base64String(Base64String&&) = default;
8      Base64String& operator=(const Base64String&) = default;
9      Base64String& operator=(Base64String&&) = default;
```

Base64String

```
10  private:
11      template <class Iterator>
12      static void throw_if_illegal(Iterator first, Iterator last) {...}
13  public:
14      template <class Iterator>
15      SpecialString& assign(const Iterator first, const Iterator last) {
16          std::for_each(first, last, Base64Char::throw_if_illegal);
17          throw_if_illegal(first, last);
18          value.resize(std::distance(first, last), ' ');
19          std::transform(first, last, value.begin(), Base64Char::create_unchecked)
20          return *this;
21      }
```

Konstruktory i operatory przypisania z std::string i **const char*** są już trywialne.

Czym jest hex?

Metoda kodowania danych binarnych w ASCII.

Czym jest hex

- Ciąg znaków z zakresu `[0-9a-f]`,
- Jeden znak to wartość z zakresu 0-15, czyli 4 bity,
- Dwa znaki to jeden bajt.

Czego wymagamy od klasy HexString?

string

- Tworzenie i przypisywanie z **const char**$*$ i std::string
  - Rzuca wyjątek, gdy dane wejściowe niepoprawne
- Iteratory i `operator[]` tylko do odczytu

Zarys implementacji

- Klasa HexChar
  - odpowiada za jeden znak,

Zarys implementacji

- Klasa HexChar
  - odpowiada za jeden znak,
- Klasa HexString
  - zawiera obiekty klasy HexChar,
  - odpowiada za cały ciąg.

Wymagania

Wygląda znajomo?

Implementacja

Mając gotową implementację Base64Char i Base64String...

```
for i in {char,string}.{c,h}pp; do
    cp base64${i} hex${i}
done
sed -i 's/Base64/Hex/g' hex*
```

Edycja HexChar::throw_if_illegal i HexString::throw_if_illegal.

DRY principle

DRY principle
(don't repeat yourself)

DRY principle
(don't repeat yourself)
Wyciągamy część wspólną

Różnice między Base64 a Hex

- {Base64,Hex}Char::throw_if_illegal
- {Base64,Hex}String::throw_if_illegal

Różnice między Base64 a Hex

- {Base64,Hex}Char::throw_if_illegal
- {Base64,Hex}String::throw_if_illegal

`virtual`?

Różnice między Base64 a Hex

- {Base64,Hex}Char::throw_if_illegal
- {Base64,Hex}String::throw_if_illegal

`virtual`? with `static`???

Metody wirtualne

```
1  struct Base {
2      virtual void bar() = 0;
3      void foo() { bar(); }
4  };
5  struct Derived1 : Base {
6      virtual void bar() override;
7  };
8  struct Derived2 : Base {
9      virtual void bar() override;
10 };
```

- Każda klasa dziedziczy po tej samej klasie Base.
- Wywoływanie metod przez `vtable`.

CRTP

CRTP
Curiously recurring template pattern

CRTP

CRTP
Curiously recurring template pattern
Klasa dziedzicząca jako parametr szablonu klasy bazowej.
class Derived : Base<Derived>

CRTP

Inaczej mówiąc – dajemy klasie bazowej dostęp do dziedziczącej.

CRTP

```
1  template <typename InheritingType>
2  struct Base {
3      void foo() { InheritingType::bar(); }
4  };
5  struct Derived1 : Base<Derived1> {
6      void bar();
7  };
8  struct Derived2 : Base<Derived2> {
9      void bar();
10 };
```

- Każda klasa dziedziczy po osobnej klasie Base<Derived>.
- Pozwala na bezpośrednie wołanie metod klasy dziedziczącej (łącznie ze statycznymi).

SpecialChar

Część trywialna.

```
1  template <class InheritingType>
2  class SpecialChar {
3      char char_;
4  public:
5      SpecialChar() = default;
6      SpecialChar(const SpecialChar&) = default;
7      SpecialChar(SpecialChar&&) = default;
8      SpecialChar& operator=(const SpecialChar&) = default;
9      SpecialChar& operator=(SpecialChar&&) = default;
10     operator char() const { return char_; }
```

SpecialChar

Rzuca wyjątek, gdy dane wejściowe niepoprawne.

```
11  public:
12      SpecialChar(char char__) : char_(char__)
13      { InheritingType::throw_if_illegal(char__); }
14
15      SpecialChar& operator=(char char__)
16      {
17          InheritingType::throw_if_illegal(char__);
18          char_ = char__;
19          return *this;
20      }
```

SpecialChar

Tylna furtka.

```
21  private:
22      static InheritingType create_unchecked(char char__) {
23          auto c = InheritingType{};
24          c.char_ = char__;
25          return c;
26      }
27  };
```

Base64Char

```
1  class Base64Char : public SpecialChar<Base64Char> {
2  private:
3      typedef SpecialChar<Base64Char> BaseType;
4      friend BaseType;
5
6      friend SpecialString<Base64String, Base64Char>;
7
8      static void throw_if_illegal(const char char_);
9  public:
10     using BaseType::BaseType;
11 };
```

SpecialString

Część trywialna.

```
1  template <class InheritingType, class SpecialCharType>
2  class SpecialString {
3  private:
4      std::basic_string<SpecialCharType> value;
5  public:
6      SpecialString() = default;
7      SpecialString(const SpecialString&) = default;
8      SpecialString(SpecialString&&) = default;
9      SpecialString& operator=(const SpecialString&) = default;
10     SpecialString& operator=(SpecialString&&) = default;
```

SpecialString

```
11  public:
12      template <class Iterator>
13      SpecialString& assign(const Iterator first, const Iterator last) {
14          std::for_each(first, last, SpecialCharType::throw_if_illegal);
15          InheritingType::throw_if_illegal(first, last);
16          value.resize(std::distance(first, last), ' ');
17          std::transform(first, last, value.begin(),
18              SpecialCharType::create_unchecked);
19          return *this;
20      }
```

Base64String

```
1  class Base64String : public SpecialString<Base64String, Base64Char> {
2  private:
3      typedef SpecialString<Base64String, Base64Char> BaseType;
4      friend BaseType;
5
6      template <Iterator>
7      static void throw_if_illegal(Iterator first, Iterator last) {...}
8  public:
9      using BaseType::BaseType;
10 };
```

HexChar i HexString

HexChar i HexString analogicznie.

Podsumowanie

```
1  struct Base {
2      virtual void bar() = 0;
3      void foo() { bar(); }
4  };
5  struct Derived1 : Base {
6      virtual void bar() override;
7  };
8  struct Derived2 : Base {
9      virtual void bar() override;
10 };
```

```
1  template <typename InheritingType>
2  struct Base {
3      void foo() { InheritingType::bar(); }
4  };
5  struct Derived1 : Base<Derived1> {
6      void bar();
7  };
8  struct Derived2 : Base<Derived2> {
9      void bar();
10 };
```

Podsumowanie

```
1  struct Base {
2      virtual void bar() = 0;
3      void foo() { bar(); }
4  };
5  struct Derived1 : Base {
6      virtual void bar() override;
7  };
8  struct Derived2 : Base {
9      virtual void bar() override;
10 };
```

```
1  template <typename InheritingType>
2  struct Base {
3      void foo() { InheritingType::bar(); }
4  };
5  struct Derived1 : Base<Derived1> {
6      void bar();
7  };
8  struct Derived2 : Base<Derived2> {
9      void bar();
10 };
```

https://github.com/etam/etam_string_types

Podsumowanie

```
1  struct Base {
2      virtual void bar() = 0;
3      void foo() { bar(); }
4  };
5  struct Derived1 : Base {
6      virtual void bar() override;
7  };
8  struct Derived2 : Base {
9      virtual void bar() override;
10 };
```

```
1  template <typename InheritingType>
2  struct Base {
3      void foo() { InheritingType::bar(); }
4  };
5  struct Derived1 : Base<Derived1> {
6      void bar();
7  };
8  struct Derived2 : Base<Derived2> {
9      void bar();
10 };
```

`https://github.com/etam/etam_string_types`

Pytania?