# Valores nulos em containers

## Uma solução genérica para usar sentinelas como valores nulos

# Problema

Como expressar nulos em um container de valores escalares?

```
vector<double> {
  30.0,
  ?? /*null*/,
  33.5
};
```

# Problema

Como expressar nulos em um container de valores escalares?

```
vector<double> {
  30.0,
  numeric_limits::quiet_NaN(),
  33.5
};
```

- Uso de NaN(not a number)

# Dificuldades

Tipo do valor não expressa a possibilidade de ausência de valor:

```
vector<double> {
  30.0,
  numeric_limits<double>::quiet_NaN(),
  33.5
};
```

# Dificuldades

Programação pouco segura para manipular o valor:

```
auto vec = vector<double> {
  30.0,
  numeric_limits<double>::quiet_NaN(),
  33.5
};
```

# Dificuldades

Programação pouco segura para manipular o valor:

```
auto vec = vector<double> {
  30.0,
  numeric_limits<double>::quiet_NaN(),
  33.5
};

//precondition: value eh um valor valido
void use_value(double value) { /*...*/ }
```

# Dificuldades

Programação pouco segura para manipular o valor:

```
auto vec = vector<double> {
  30.0,
  numeric_limits<double>::quiet_NaN(),
  33.5
};

//precondition: value eh um valor valido
void use_value(double value) { /*...*/ }

use_value(vec[i]);
```

# Dificuldades

Programação pouco segura para manipular o valor:

```
auto vec = vector<double> {
  30.0,
  numeric_limits<double>::quiet_NaN(),
  33.5
};

//precondition: value eh um valor valido
void use_value(double value) { /*...*/ }

use_value(vec[i]); //Bug em runtime
```

# Dificuldades

Programação pouco segura para manipular o valor:

```
auto vec = vector<double> {
  30.0,
  numeric_limits<double>::quiet_NaN(),
  33.5
};

//precondition: value eh um valor valido
void use_value(double value) { /*...*/ }

if (!isnan(vec[i]))
  use_value(vec[i]);
```

# Dificuldades

Programação pouco segura para manipular o valor:

```
auto vec = vector<double> {
  30.0,
  numeric_limits<double>::quiet_NaN(),
  33.5
};

//precondition: value eh um valor valido
void use_value(double value) { /*...*/ }

if (!isnan(vec[i]))
  use_value(vec[i]);
```

- O compilador não me ajuda, é necessário lembrar uso de isnan()

# Dificuldades

- Não existe NaN para integrais

# Dificuldades

- Não existe NaN para integrais
- É necessário escolher um valor válido do conjunto de valores

# Dificuldades

- Não existe NaN para integrais
- É necessário escolher um valor válido do conjunto de valores
- O valor escolhido pode não funcionar para todos os contextos do software

# Dificuldades

- Não existe NaN para integrais
- É necessário escolher um valor válido do conjunto de valores
- O valor escolhido pode não funcionar para todos os contextos do software

```
auto vec_1 = vector<unsigned char> {
  145,
  numeric_limits<unsigned char>::max(),
  50
};
```

# Dificuldades

- Não existe NaN para integrais
- É necessário escolher um valor válido do conjunto de valores
- O valor escolhido pode não funcionar para todos os contextos do software

```
auto vec_1 = vector<unsigned char> {
  145,
  numeric_limits<unsigned char>::max(),
  50
};

auto vec_2 = vector<unsigned char> {
  255,
  numeric_limits<unsigned char>::min(),
  89
};
```

# Dificuldades

Posso misturar valores nulos diferentes de containers do mesmo tipo:

```
//precondition: value eh um valor valido
void use_value(unsigned char value) { /*...*/ }

auto null = numeric_limits<unsigned char>::max();
if (vec_1[i] != null)
  use_value(vec_1[i])
```

# Dificuldades

Posso misturar valores nulos diferentes de containers do mesmo tipo:

```
//precondition: value eh um valor valido
void use_value(unsigned char value) { /*...*/ }

auto null = numeric_limits<unsigned char>::min();
if (vec_1[i] != null)
  use_value(vec_1[i])
```

# Dificuldades

Posso misturar valores nulos diferentes de containers do mesmo tipo:

```
//precondition: value eh um valor valido
void use_value(unsigned char value) { /*...*/ }

auto null = numeric_limits<unsigned char>::min();
if (vec_1[i] != null)
  use_value(vec_1[i]) //Bug em runtime
```

# Dificuldades

- Baixa expressividade

# Dificuldades

- Baixa expressividade
- O compilador não ajuda a verificar nulos

# Dificuldades

- Baixa expressividade
- O compilador não ajuda a verificar nulos
- Programador pode misturar valores nulos

# Boost.Optional

- Expressa no tipo a possibilidade de ausência de valor

```
auto vec_1 = vector<boost::optional<double>> {
  30.0,
  {},
  33.5
};
```

# Boost.Optional

- Expressa no tipo a possibilidade de ausência de valor

```
auto vec_1 = vector<boost::optional<double>> {
   30.0,
   {},
   33.5
};

use_value(vec_1[i]);
```

# Boost.Optional

- Expressa no tipo a possibilidade de ausência de valor

```
auto vec_1 = vector<boost::optional<double>> {
   30.0,
   {},
   33.5
};

use_value(vec_1[i]);  //Ótimo. Erro de compilação.
```

- Compilador notifica que o nulo deve ser tratado

# Boost.Optional

- Expressa no tipo a possibilidade de ausência de valor

```
auto vec_1 = vector<boost::optional<double>> {
   30.0,
   {},
   33.5
};
```

```
if (vec_1[i]) use_value(*vec_1[i]);
```

# Boost.Optional

- Expressa no tipo a possibilidade de ausência de valor

```
auto vec_1 = vector<boost::optional<double>> {
  30.0,
  {},
  33.5
};
```

```
if (vec_1[i]) use_value(*vec_1[i]);
```

- Não é necessário lidar com valores especiais(NaN, INT_MAX, ...)

# Boost.Optional

- Ineficiência(no espaço) inviável para container com muitos elementos

```
sizeof(boost::optional<double>>) = 2*sizeof(double>)
```

# Boost.Optional

- Ineficiência(no espaço) inviável para container com muitos elementos

```
sizeof(boost::optional<double>>) = 2*sizeof(double>)
```

- Se double é 8 bytes então boost::optional<double> é 16 bytes!

# Boost.Optional

- Ineficiência(no espaço) inviável para container com muitos elementos

```
sizeof(boost::optional<double>>) = 2*sizeof(double>)
```

- Se double é 8 bytes então boost::optional<double> é 16 bytes!
- Compilador alinha a memória da estrutura definida, por exemplo:

# Boost.Optional

- Ineficiência(no espaço) inviável para container com muitos elementos

```
sizeof(boost::optional<double>>) = 2*sizeof(double>)
```

- Se double é 8 bytes então boost::optional<double> é 16 bytes!
- Compilador alinha a memória da estrutura definida, por exemplo:

```
struct optional_double {
  double value;
  bool _initialized;
};
```

# Boost.Optional

- Ineficiência(no espaço) inviável para container com muitos elementos

```
sizeof(boost::optional<double>>) = 2*sizeof(double>)
```

- Se double é 8 bytes então boost::optional<double> é 16 bytes!
- Compilador alinha a memória da estrutura definida, por exemplo:

```
struct optional_double {
  double value; //8 bytes
  bool _initialized;
};
```

# Boost.Optional

- Ineficiência(no espaço) inviável para container com muitos elementos

```
sizeof(boost::optional<double>) = 2*sizeof(double)
```

- Se double é 8 bytes então boost::optional<double> é 16 bytes!
- Compilador alinha a memória da estrutura definida, por exemplo:

```
struct optional_double {
  double value; //8 bytes
  bool _initialized; //1 byte
};
```

# Boost.Optional

- Ineficiência(no espaço) inviável para container com muitos elementos

```
sizeof(boost::optional<double>) = 2*sizeof(double)
```

- Se double é 8 bytes então boost::optional<double> é 16 bytes!
- Compilador alinha a memória da estrutura definida, por exemplo:

```
struct optional_double {
  double value; //8 bytes
  bool _initialized; //1 byte
  char _pad[7]; //7 bytes
};
```

# Nullable(ak_toolkit::markable)

Um “optional” com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable

# Nullable(ak_toolkit::markable)

Um “optional” com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable
  - Tentativa de um nome mais expressivo

# Nullable(ak_toolkit::markable)

Um “optional” com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable
  - Tentativa de um nome mais expressivo
  - Define valores nulos padrões para integrais e ponto flutuante: numeric_limits<Int>:max() e NaN.

# Nullable(ak_toolkit::markable)

Um “optional” com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable
  - Tentativa de um nome mais expressivo
  - Define valores nulos padrões para integrais e ponto flutuante: numeric_limits<Int>:max() e NaN.
- Expressividade do boost::optional

# Nullable(ak_toolkit::markable)

Um “optional” com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable
  - Tentativa de um nome mais expressivo
  - Define valores nulos padrões para integrais e ponto flutuante: numeric_limits<Int>:max() e NaN.
- Expressividade do boost::optional
- Segurança no uso como no boost::optional

# Nullable(ak_toolkit::markable)

Um “optional” com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable
  - Tentativa de um nome mais expressivo
  - Define valores nulos padrões para integrais e ponto flutuante: numeric_limits<Int>:max() e NaN.
- Expressividade do boost::optional
- Segurança no uso como no boost::optional
- Eficiente: sizeof(Nullable<T>) == sizeof(T)

# Nullable(ak_toolkit::markable)

Um "optional" com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable
  - Tentativa de um nome mais expressivo
  - Define valores nulos padrões para integrais e ponto flutuante: numeric_limits<Int>:max() e NaN.
- Expressividade do boost::optional
- Segurança no uso como no boost::optional
- Eficiente: sizeof(Nullable<T>) == sizeof(T)

```
auto vec = vector<Nullable<double>> {
  30.0,
  Nullable<double>{},
  33.5
};
```

# Nullable(ak_toolkit::markable)

Um “optional” com valor nulo definido em compile time

- Nullable é somente uma conveniência para ak_toolkit::markable
  - Tentativa de um nome mais expressivo
  - Define valores nulos padrões para integrais e ponto flutuante: numeric_limits<Int>:max() e NaN.
- Expressividade do boost::optional
- Segurança no uso como no boost::optional
- Eficiente: sizeof(Nullable<T>) == sizeof(T)

```
auto vec = vector<Nullable<double>> {
  30.0,
  Nullable<double>{},
  33.5
};

if (vec[i].has_value())
  use_value(vec[i].value());
```

# Construção com nulos definidos em runtime

Construção do container a partir de fonte externa que utiliza valor nulo definido em runtime

- Não sabemos qual é o valor nulo quando escrevemos o software

# Construção com nulos definidos em runtime

Construção do container a partir de fonte externa que utiliza valor nulo definido em runtime

- Não sabemos qual é o valor nulo quando escrevemos o software
- Após importação o valor nulo em runtime não é mais necessário para o container

# Construção com nulos definidos em runtime

Construção do container a partir de fonte externa que utiliza valor nulo definido em runtime

- Não sabemos qual é o valor nulo quando escrevemos o software
- Após importação o valor nulo em runtime não é mais necessário para o container
- Conveniência toNullable(value, null) para construir Nullable<T>

# Construção com nulos definidos em runtime

Construção do container a partir de fonte externa que utiliza valor nulo definido em runtime

- Não sabemos qual é o valor nulo quando escrevemos o software
- Após importação o valor nulo em runtime não é mais necessário para o container
- Conveniência toNullable(value, null) para construir Nullable<T>
  - Evita código boilerplate de verificação de nulo

# Construção com nulos definidos em runtime

Construção do container a partir de fonte externa que utiliza valor nulo definido em runtime

- Não sabemos qual é o valor nulo quando escrevemos o software
- Após importação o valor nulo em runtime não é mais necessário para o container
- Conveniência toNullable(value, null) para construir Nullable<T>
  - Evita código boilerplate de verificação de nulo

```
(null == value) ? Nullable<T>() : Nullable<T>(value)
```

# Construção com nulos definidos em runtime

Construção do container a partir de fonte externa que utiliza valor nulo definido em runtime

- Não sabemos qual é o valor nulo quando escrevemos o software
- Após importação o valor nulo em runtime não é mais necessário para o container
- Conveniência toNullable(value, null) para construir Nullable<T>
  - Evita código boilerplate de verificação de nulo

```
(null == value) ? Nullable<T>() : Nullable<T>(value)
```

```
double null = -999.25;
vec.push_back(toNullable(value, null));
```

# [Extra] Sentinelas customizadas (MarkPolicy)

Uso de uma policy customizada MarkPolicy para usar um valor de sentinela customizado:

- Permite por exemplo definir valores de mesmo tipo com valores nulos distintos

# [Extra] Sentinelas customizadas (MarkPolicy)

Uso de uma policy customizada MarkPolicy para usar um valor de sentinela customizado:

- Permite por exemplo definir valores de mesmo tipo com valores nulos distintos

```
using NullableIntMin = Nullable<
  int,
  mark_int<int, numeric_limits<int>::min()>
>;

using NullableIntMax = Nullable<
  int,
  mark_int<int, numeric_limits<int>::max()>
>;
```

# Referências

- Boost.Optional
  `http://www.boost.org/doc/libs/release/libs/optional`
- `https://github.com/akrzemi1/markable`