Programovacie techniky

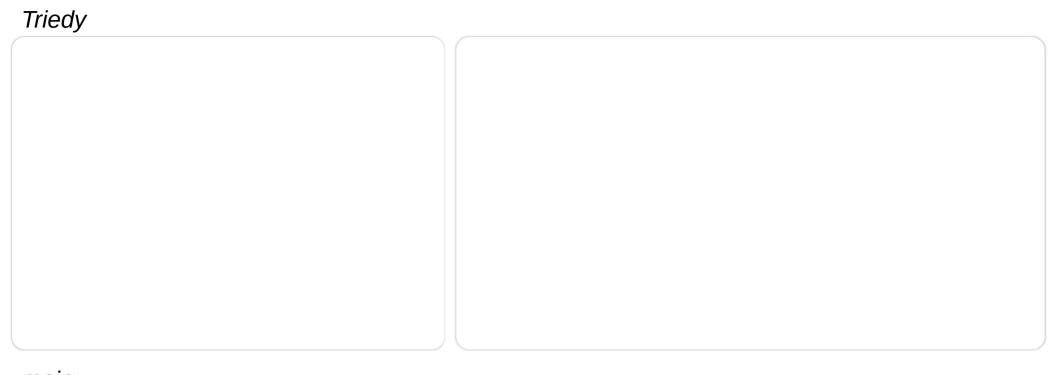
Pretažovanie operátorov v C++

Vladislav Novák

Obsah

```
operator[] a[5]
operator* * it
operator++ it ++
operator!= a != b
operator bool() if (a) bool(a)

explicit - umožnuje len explicitné pretypovanie (zakazuje implicitné pretypovanie)
```

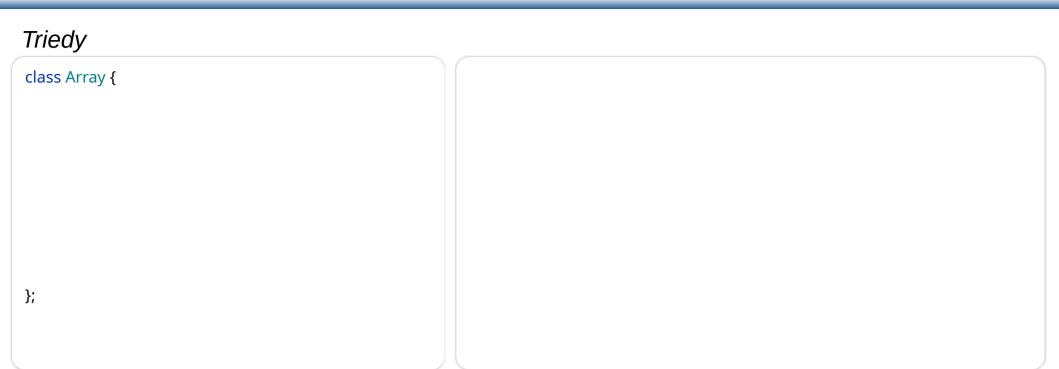


```
Array array(5);

size_t size = array.getSize();

int value = array.get(0);

array.set(0, 100);
```



```
Array array(5);

size_t size = array.getSize();

int value = array.get(0);

array.set(0, 100);
```

Triedy

```
class Array {
private:
  int *data;
  const size_t size;
};
```

```
Array array(5);

size_t size = array.getSize();

int value = array.get(0);

array.set(0, 100);
```

Triedy

```
class Array {
  private:
    int *data;
    const size_t size;
  public:
    explicit Array(size_t size);
    ~Array();
    size_t getSize() const noexcept;
    int get(size_t index) const;
    void set(size_t index, int value);
};
```

```
Array array(5);

size_t size = array.getSize();

int value = array.get(0);

array.set(0, 100);
```

Triedy

```
class Array {
private:
   int *data;
   const size_t size;
public:
   explicit Array(size_t size);
   ~Array();
   size_t getSize() const noexcept;
   //?
};
```

```
Array array(5);
size_t size = array.getSize();
int value = array[0];
```

Triedy

```
class Array {
  private:
    int *data;
    const size_t size;
  public:
    explicit Array(size_t size);
    ~Array();
    size_t getSize() const noexcept;
    int operator[](size_t index);
};
```

```
Array array(5);
size_t size = array.getSize();
int value = array[0];
```

Triedy

```
class Array {
private:
  int *data;
  const size_t size;
public:
  explicit Array(size_t size);
  ~Array();
  size_t getSize() const noexcept;
  int & operator[](size_t index);
};
```

```
Array array(5);

size_t size = array.getSize();

int value = array[0];

array[0] = 100;
```

Triedy

```
class Array {
private:
   int *data;
   const size_t size;
public:
   explicit Array(size_t size);
   ~Array();
   size_t getSize() const noexcept;
   int & operator[](size_t index);
};
```

```
Array::Array(size_t size) : size(size) {
  data = new int [size];
}
```

```
Array array(5);

size_t size = array.getSize();

int value = array[0];

array[0] = 100;
```

Triedy

```
class Array {
private:
   int *data;
   const size_t size;
public:
   explicit Array(size_t size);
   ~Array();
   size_t getSize() const noexcept;
   int & operator[](size_t index);
};
```

```
Array::Array(size_t size) : size(size) {
   data = new int [size];
}
Array::~Array() {
   delete[] data;
}
```

```
Array array(5);

size_t size = array.getSize();

int value = array[0];

array[0] = 100;
```

Triedy

```
class Array {
private:
   int *data;
   const size_t size;
public:
   explicit Array(size_t size);
   ~Array();
   size_t getSize() const noexcept;
   int & operator[](size_t index);
};
```

```
Array::Array(size_t size) : size(size) {
    data = new int [size];
}
Array::~Array() {
    delete[] data;
}
size_t Array::getSize() const noexcept {
    return size;
}
```

```
Array array(5);

size_t size = array.getSize();

int value = array[0];

array[0] = 100;
```

Triedy

```
class Array {
private:
   int *data;
   const size_t size;
public:
   explicit Array(size_t size);
   ~Array();
   size_t getSize() const noexcept;
   int & operator[](size_t index);
};
```

```
Array array(5);

size_t size = array.getSize();

int value = array[0];

array[0] = 100;
```

```
Array::Array(size_t size) : size(size) {
  data = new int [size];
Array::~Array() {
  delete∏ data;
size_t Array::getSize() const noexcept {
  return size:
int & Array::operator[](size_t index) {
  if(index < size) {</pre>
    return data[index];
  else {
    throw out_of_range(to_string(index));
```

Triedy

```
class Array {
private:
  int *data;
  const size_t size;
public:
  explicit Array(size_t size);
  ~Array();
  size_t getSize() const noexcept;
  int & operator[](size_t index);
};
```

Triedy

```
class Array {
private:
  int *data;
  const size_t size;
public:
  explicit Array(size_t size);
  ~Array();
  size_t getSize() const noexcept;
  int & operator[](size_t index);
  //?
};
```

```
Array array(5);

for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
   cout << (*it) << endl;
}</pre>
```

Triedy

```
class Array {
private:
  int *data;
  const size_t size;
public:
  explicit Array(size_t size);
  ~Array();
  size_t getSize() const noexcept;
  int & operator[](size_t index);
  //?
};
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

Triedy

```
class Array {
private:
  int *data;
  const size_t size;
public:
  explicit Array(size_t size);
  ~Array();
  size_t getSize() const noexcept;
  int & operator[](size_t index);
  class Iterator;
  //?
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

```
Triedy
```

```
class Array {
private:
  int *data;
  const size_t size;
public:
  explicit Array(size_t size);
  ~Array();
  size_t getSize() const noexcept;
  int & operator[](size_t index);
  class Iterator;
  Iterator begin();
  Iterator end();
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

Triedy

```
class Array {
private:
  int *data;
  const size_t size;
public:
  explicit Array(size_t size);
  ~Array();
  size_t getSize() const noexcept;
  int & operator[](size_t index);
  class Iterator;
  Iterator begin();
  Iterator end();
```

```
Array::Iterator Array::begin() {
    return Iterator(this);
}
Array::Iterator Array::end() {
    return Iterator(this, size);
}
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

```
Triedy
```

```
Array::Iterator Array::begin() {
    return Iterator(this);
}
Array::Iterator Array::end() {
    return Iterator(this, size);
}
```

```
Array array(5);

for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
   cout << (*it) << endl;
}</pre>
```

```
Array array(5);

for(int element : array) {
   cout << element << endl;
}
```

Triedy

```
class Array::Iterator {
private:
Array *array;
size_t index;
```

```
Array::Iterator Array::begin() {
    return Iterator(this);
}
Array::Iterator Array::end() {
    return Iterator(this, size);
}
```

main:

};

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);

for(int element : array) {
   cout << element << endl;
}
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size_t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
};
```

```
Array::Iterator Array::begin() {
    return Iterator(this);
}
Array::Iterator Array::end() {
    return Iterator(this, size);
}
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size_t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
};
```

```
Array::Iterator::Iterator(Array * array, size_t index)
: array(array)
, index(index) {
}
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size_t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
};
```

```
Array::Iterator(Array * array, size_t index)
: array(array)
, index(index) {
}
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);

for(int element : array) {
   cout << element << endl;
}
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size_t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
};
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size_t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
  Iterator & operator++() noexcept; // prefixovy
};
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
   cout << element << endl;
}</pre>
```

Triedy

```
class Array::Iterator {
  private:
    Array *array;
    size_t index;
  public:
    explicit Iterator(Array * array, size_t index = 0);
    int & operator*() const;
    Iterator & operator++() noexcept; // prefixovy
};
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
  cout << element << endl;
}</pre>
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
  Iterator & operator++() noexcept; // prefixovy
 //Iterator operator++(int) noexcept; // postfixovy
};
```

```
Array::Iterator::Iterator(Array * array, size_t index)
    : array(array)
    , index(index) {
}
int & Array::Iterator::operator*() const {
    return (*array)[index];
}
Array::Iterator & Array::Iterator::operator++() noexcept {
    ++ index;
    return *this;
}
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);

for(int element : array) {
   cout << element << endl;
}
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
  Iterator & operator++() noexcept; // prefixovy
 //Iterator operator++(int) noexcept; // postfixovy
};
```

```
Array::Iterator::Iterator(Array * array, size_t index)
  : array(array)
  , index(index) {
int & Array::Iterator::operator*() const {
  return (*array)[index];
Array::Iterator & Array::Iterator::operator++() noexcept {
  ++ index:
  return *this:
//Array::Iterator Array::Iterator::operator++(int) noexcept {
   Iterator previous(*this);
    operator++();
   return previous;
//}
```

```
Array array(5);

for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
   cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
  cout << element << endl;
}</pre>
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
  Iterator & operator++() noexcept; // prefixovy
 //Iterator operator++(int) noexcept; // postfixovy
};
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
  cout << element << endl;
}</pre>
```

Triedy

```
class Array::Iterator {
private:
  Array *array;
  size t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
  Iterator & operator++() noexcept; // prefixovy
 //Iterator operator++(int) noexcept; // postfixovy
  bool operator!=(const Iterator &other) const
                                     noexcept;
};
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

```
Array array(5);
for(int element : array) {
  cout << element << endl;
}</pre>
```

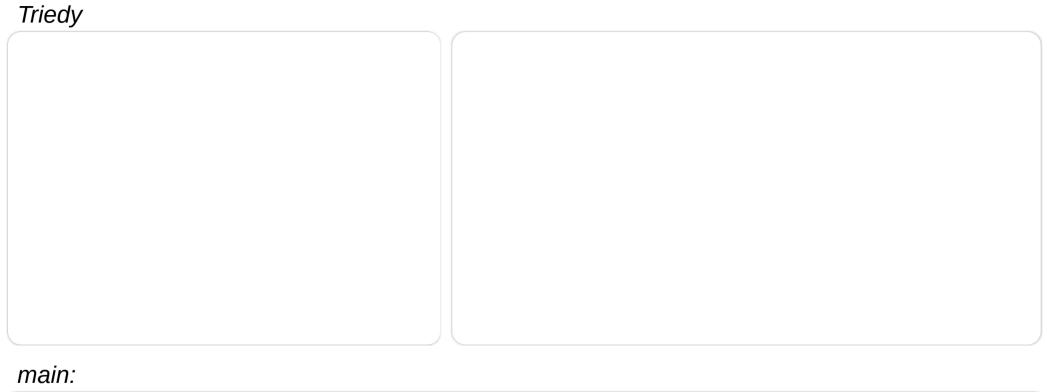
Triedy

```
class Array::Iterator {
private:
  Array *array;
  size t index;
public:
  explicit Iterator(Array * array, size_t index = 0);
  int & operator*() const;
  Iterator & operator++() noexcept; // prefixovy
 //Iterator operator++(int) noexcept; // postfixovy
  bool operator!=(const Iterator &other) const
                                     noexcept;
};
```

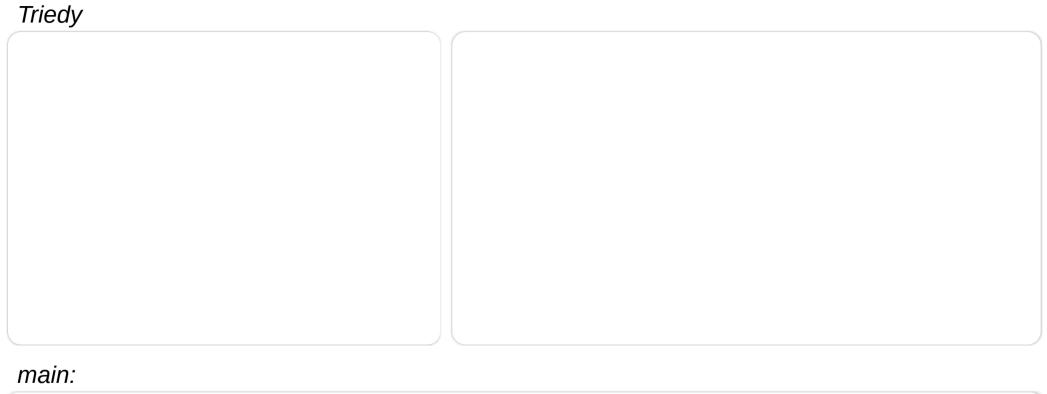
```
Array::Iterator::Iterator(Array * array, size t index)
  : array(array)
  , index(index) {
int & Array::Iterator::operator*() const {
  return (*array)[index];
Array::Iterator & Array::Iterator::operator++() noexcept {
  ++ index:
  return *this:
bool Array::Iterator::operator!=(const Array::Iterator &other) const
                                                     noexcept {
  return this->array != other.array || this->index != other.index;
```

```
Array array(5);
for(Array::Iterator it = array.begin(); it != array.end(); ++it) {
  cout << (*it) << endl;
}</pre>
```

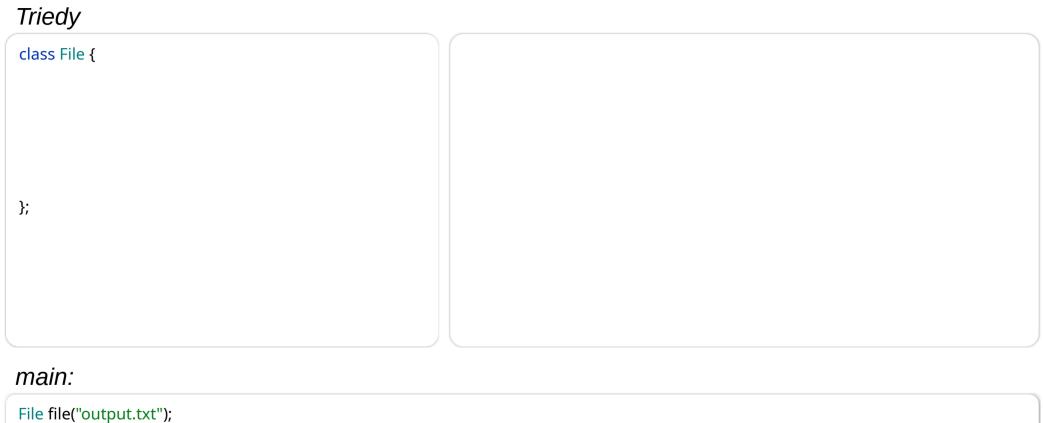
```
Array array(5);
for(int element : array) {
  cout << element << endl;
}</pre>
```



```
File file("output.txt");
if(file) {
```



```
File file("output.txt");
if(! file) {
   // .....
```



if(! file) { //

```
Triedy
```

```
class File {
private:
  const string name;
```

```
main:
File file("output.txt");
if(! file) {
   // .....
```

Triedy

```
class File {
private:
  const string name;
public:
  explicit File(const string & name);
main:
```

mani

Triedy

```
class File {
  private:
    const string name;
  public:
    explicit File(const string & name);
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
```

Triedy

```
class File {
private:
   const string name;
public:
   explicit File(const string & name);
   explicit operator bool() const noexcept;
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}
```

Triedy

```
class File {
  private:
    const string name;
  public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}
```

```
File file("output.txt");

if(! file) { // v podmienke môžeme použiť aj s deklaráciou operátora ako explicit
// .....
}
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}
```

```
File file("output.txt");
```

Triedy

```
class File {
    private:
        const string name;
    public:
        explicit File(const string & name);
        explicit operator bool() const noexcept;
        // ďalšie metódy, deštruktor
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}
```

main:

File file("output.txt");

```
bool value = file; // chyba, musime explicitne pretypovat, alebo vymazat explicit
```

Triedy

```
class File {
    private:
        const string name;
    public:
        explicit File(const string & name);
        explicit operator bool() const noexcept;
        // ďalšie metódy, deštruktor
};
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}
```

main:

```
bool value = bool(file); // ok
```

File file("output.txt");

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};

void fn(bool value);
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}

void fn(bool value) {
    // .....
}
```

```
File file("output.txt");
bool value = bool(file); // ok
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};

void fn(bool value);
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}

void fn(bool value) {
    // .....
}
```

```
File file("output.txt");

bool value = bool(file); // ok

fn(file); // chyba, musime explicitne pretypovat, alebo vymazat explicit
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};

void fn(bool value);
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}

void fn(bool value) {
    // .....
}
```

```
File file("output.txt");

bool value = bool(file); // ok

fn(bool(file)); // ok
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};
void fn(bool value);
```

```
File::File(const string & name): name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}

void fn(bool value) {
    // .....
}
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};

void fn(bool value);
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}

void fn(bool value) {
    // .....
}
```

```
File file1 = string("output1.txt"); // chyba, kompilátor nerobí explicitné pretypovanie
```

Triedy

```
class File {
private:
    const string name;
public:
    explicit File(const string & name);
    explicit operator bool() const noexcept;
    // d'alšie metódy, deštruktor
};

void fn(bool value);
```

```
File::File(const string & name) : name(name) {
    /* otvorenie suboru */
}
File::operator bool() const noexcept {
    return true/false;
}

void fn(bool value) {
    // .....
}
```

```
File file1 = string("output1.txt"); // chyba, kompilátor nerobí explicitné pretypovanie

string name2 = "output2.txt";

File file2 = name2; // chyba, kompilátor nerobí explicitné pretypovanie
```