

C++ don't do it at home

Robert Piszczatowski

Tls technologie (tls.pl)

DCC Labs (dcclabs.com)

Warsaw C++ Users Group, 2015

Ternary operator

Not so obvious...

Ternary operator return type.

Ternary operator

Not so obvious...

Ternary operator return type.

```
1 A a1, a2;  
2 bool b = ...;  
3 (b ? a1 : a2).f();
```

Ternary operator

Not so obvious...

Ternary operator return type.

```
1 A a1, a2;  
2 bool b = ...;  
3 (b ? a1 : a2).f();  
4 (b ? a1 : A()).f();
```

Ternary operator

Not so obvious...

Ternary operator return type.

```
1 A a1, a2;  
2 bool b = ...;  
3 (b ? a1 : a2).f();  
4 (b ? a1 : A()).f(); // creates a copy of a1
```

Ternary operator

Not so obvious...

Ternary operator return type.

```
1 A a1, a2;
2 bool b = ...;
3 (b ? a1 : a2).f();
4 (b ? a1 : A()).f(); // creates a copy of a1
```

Ternary operator in PHP is left associative

```
1 TRUE ? "a" : FALSE ? "b" : "c"; # this evaluates to "b"
```

Ternary operator

Not so obvious...

Ternary operator return type.

```
1 A a1, a2;
2 bool b = ...;
3 (b ? a1 : a2).f();
4 (b ? a1 : A()).f(); // creates a copy of a1
```

Ternary operator in PHP is left associative

```
1 TRUE ? "a" : FALSE ? "b" : "c"; # this evaluates to "b"
```

Ternary operator precedence

```
1 a = b ? c : d = e
2
```

Ternary operator

Not so obvious...

Ternary operator return type.

```
1 A a1, a2;
2 bool b = ...;
3 (b ? a1 : a2).f();
4 (b ? a1 : A()).f(); // creates a copy of a1
```

Ternary operator in PHP is left associative

```
1 TRUE ? "a" : FALSE ? "b" : "c"; # this evaluates to "b"
```

Ternary operator precedence

```
1 a = b ? c : d = e
2 a = (b ? c : (d = e)) // same as above in c++
3
```


Ternary operator

Not so obvious...

Ternary operator return type.

```
1 A a1, a2;
2 bool b = ...;
3 (b ? a1 : a2).f();
4 (b ? a1 : A()).f(); // creates a copy of a1
```

Ternary operator in PHP is left associative

```
1 TRUE ? "a" : FALSE ? "b" : "c"; # this evaluates to "b"
```

Ternary operator precedence

```
1 a = b ? c : d = e
2 a = (b ? c : (d = e)) // same as above in c++
3 a = (b ? c : d) = e // same as above in c
```

Ternary operator

Not so obvious...

C++14 standard (n3979), §5.16:

conditional-expression:

logical-or-expression

logical-or-expression ? expression : **assignment**-expression

Ternary operator

Not so obvious...

C++14 standard (n3979), §5.16:

conditional-expression:

logical-or-expression

logical-or-expression ? expression : **assignment**-expression

C11 standard (n1570), §6.5.15:

conditional-expression:

logical-OR-expression

logical-OR-expression ? expression : **conditional**-expression

Statement expression (gcc only)

One day I saw piece of code like this below

```
1 bool b = ...;  
2 b ? (LOG_DEBUG(...)) : (LOG_DEBUG(...)); // ???
```

Statement expression (gcc only)

One day I saw piece of code like this below

```
1 bool b = ...;
2 b ? (LOG_DEBUG(...)) : (LOG_DEBUG(...)); // ???
```

This feature is called statement expressions. Standard example:

```
1 #define MY_MAX(a, b) ({int _a = (a), _b = (b); _a > _b
   ? _a : _b; })
```

Statement expression (gcc only)

One day I saw piece of code like this below

```
1 bool b = ...;
2 b ? (LOG_DEBUG(...)) : (LOG_DEBUG(...)); // ???
```

This feature is called statement expressions. Standard example:

```
1 #define MY_MAX(a, b) ({int _a = (a), _b = (b); _a > _b
   ? _a : _b; })
```

Example from Qt library

```
1 #define Q_FOREACH(variable, container) \
2 for (QForeachContainer<__typeof__(container)> _container_(container); \
3     !_container_.brk && _container_.i != _container_.e; \
4     __extension__ ({ ++_container_.brk; ++_container_.i; })) \
5 for (variable = *_container_.i; __extension__ ({--_container_.brk; break;});)
```

Void

unusable type?

It is possible to write `void()` as an expression. How it could be useful?

Void

unusable type?

It is possible to write `void()` as an expression. How it could be useful?

```
1 #define CHECK_INIT(retVal) if (!isInitialized()) return
   retVal;
2
3 void myFun() {
4     CHECK_INIT(void());
5     // do stuff...
6 }
```

Void

unusable type?

It is possible to write `void()` as an expression. How it could be useful?

```
1 #define CHECK_INIT(retVal) if (!isInitialized()) return
   retVal;
2
3 void myFun() {
4     CHECK_INIT(void());
5     // do stuff...
6 }
```

Void type can be used in argument list. In c++ it has same meaning as empty argument list.

```
1 int f(); // this can get any number of arguments in c
2 int f(void); // this can get no arguments in c
```

Void

unusable type?

According to §3.9.1/9 in c++ standard: „Any expression can be explicitly converted to type cv void.” It can be useful to explicitly ignore result of an expression.

Void

unusable type?

According to §3.9.1/9 in c++ standard: „Any expression can be explicitly converted to type cv void.” It can be useful to explicitly ignore result of an expression. But for classes user can define his own conversion. Consider:

```
1 struct A {
2     operator void() { /* some bad stuff here */ }
3 };
```

Void

unusable type?

According to §3.9.1/9 in c++ standard: „Any expression can be explicitly converted to type cv void.” It can be useful to explicitly ignore result of an expression. But for classes user can define his own conversion. Consider:

```
1 struct A {
2     operator void() { /* some bad stuff here */ }
3 };
```

According to §12.3.2/1: „A conversion function is never used to convert an object to the same object type, to a base class of that type, or to void.” So above cast operator can be used only in function call notation:

```
1 A a;
2 (void) a; // this calls builtin conversion to void
3 a.operator void(); // this calls user defined
   conversion operator
```

Function that cannot be called

There is a way to define a function that cannot be called, because there is no such syntax in language to call it.

Function that cannot be called

There is a way to define a function that cannot be called, because there is no such syntax in language to call it.

```
1 struct A {
2     template <typename T> A() {}
3     template <typename T> operator typename T::type() {}
4 };
```

Function that cannot be called

There is a way to define a function that cannot be called, because there is no such syntax in language to call it.

```
1 struct A {  
2     template <typename T> A() {}  
3     template <typename T> operator typename T::type() {}  
4 };
```

The idea here is that constructors and cast operators does not have names, so we need to create template for which template arguments will not be deductible. See §14.5.2/5: „Note: Because the explicit template argument list follows the function template name, and because conversion member function templates and constructor member function templates are called without using a function name, there is no way to provide an explicit template argument list for these function templates.”

Complicated code

```
1 int main(int t, int u, char *a) {
2     return (!0<t)?((t<3?main(-79,-13,a+main(-87,1-u,main(
3     -86,0,a+1)+a)):1),(t<u?main(t+1,u,a):3),(main(-94,-27+t,a)&&(t==2?
4     (u<13?main(2,u+1,"%s %d %d\n"):9) :16))):(t<0?(t<-72?main(u, t,
5     "@n'+,#/*}{w+/w#cdnr/+,}{r/*de}+,*{*+,/w{%+,/w#q#n+,#{[,+\\
6     ,/n{n+,/+#n+,#;#q#n+,/+#k#;*,/'r  : 'd*'3},{w+K w'K:'+'e#';d\\
7     q#'l q#'d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl}'/#;#q#n')}{#}w')}{nl ]\\
8     '/+#n';d}rw' i;# )}{nl ]!/ n{n#'; r{#w'r nc{nl ]}'/# l ,+'K {rw' iK {;[\\
9     {nl ]/' w#q#n'wk nw' iwK{KK{nl]!/w{%l##w#' i; :{nl ]}'/* q#l'd; r '}\\
10    nlwb!/*de}'c ;;{nl ]-}{rw ]'/+,}##*})# nc ,#nw ]'/+\\
11    kd'+e}+;#rdq#w! nr'/ ' ) }+}{rl #'{n' ' )# }'+}##(!/ "):( t<-50?(u==*a?
12    putchar(a[31]):
13    main(-65,u,a+1)):main((*a=='/")+t,u,a+1))):(0<t? main(2,2,"%s")
14    :*a=='/' || main(0,main(-61,*a,"lek;dc
15    i@bK'(q)-[w]*%n+r3#!,}{:\\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1))));
16 }
```


Complicated code (result)

On the first day of Christmas my true love gave to me a partridge in a pear tree.
On the second day of Christmas my true love gave to me two turtle doves and a partridge in a pear tree.
On the third day of Christmas my true love gave to me three french hens, two turtle doves and a partridge in a pear tree.
On the fourth day of Christmas my true love gave to me four calling birds, three french hens, two turtle doves and a partridge in a pear tree.
On the fifth day of Christmas my true love gave to me five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.
On the sixth day of Christmas my true love gave to me six geese a-laying, five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.
On the seventh day of Christmas my true love gave to me seven swans a-swimming, six geese a-laying, five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.
On the eighth day of Christmas my true love gave to me eight maids a-milking, seven swans a-swimming, six geese a-laying, five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.
On the ninth day of Christmas my true love gave to me nine ladies dancing, eight maids a-milking, seven swans a-swimming, six geese a-laying, five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.
On the tenth day of Christmas my true love gave to me ten lords a-leaping, nine ladies dancing, eight maids a-milking, seven swans a-swimming, six geese a-laying, five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.
On the eleventh day of Christmas my true love gave to me eleven pipers piping, ten lords a-leaping, nine ladies dancing, eight maids a-milking, seven swans a-swimming, six geese a-laying, five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.

On the twelfth day of Christmas my true love gave to me twelve drummers drumming, eleven pipers piping, ten lords a-leaping, nine ladies dancing, eight maids a-milking, seven swans a-swimming, six geese a-laying, five gold rings; four calling birds, three french hens, two turtle doves and a partridge in a pear tree.

Thank you

Thank You