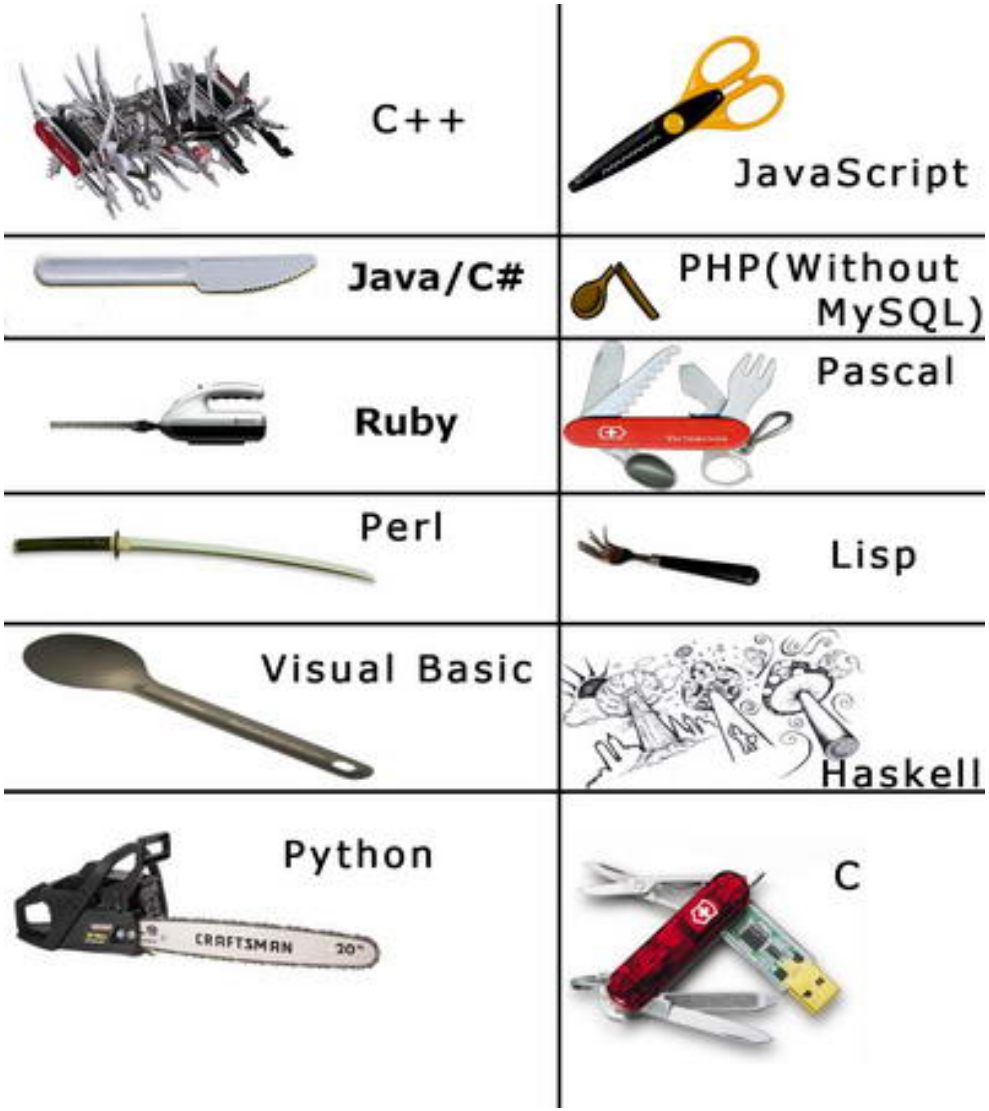
Operátorok, kifejezések, kiértékelés

Pataki Norbert

ELTE Informatikai Kar,
Programozási Nyelvek és
Fordítóprogramok Tanszék

2014. október 3.

Programozási nyelvek



Vizsgaidőpontok

- 2014. december 19. 14:00, Lovarda
- 2015. január 5. 14:00, Lovarda
- 2015. január 12. 14:00, Lovarda
- 2015. január 19. 14:00, Lovarda
- 2015. január 26. 14:00, Lovarda

Tokenek

- Kulcsszavak
- Azonosítók
- Konstansok
- Konstans szövegliterálok
- Operátorok
- Szeparátorok

Tulajdonságok

- Precedencia
- Kötés, asszociáció
- `x = 3 + 5 * b;`
- `x = (3 + (5 * b));`
- `x = (3 + 5) * b;`
- `a = b = c = d;`
- `(a = (b = (c = d)));`

Operátorok

Precedence and associativity of C++ operators

Operator	Description	Associativity
::	Global scope resolution	Right to left
::	Class scope resolution	Left to right
( )	Function call	Left to right
[ ]	Array subscript	Left to right
->	Indirect member selector	Left to right
.	Direct member selector	Left to right
++	Post increment	Left to right
--	Post decrement	Left to right
!	Logical negation	Right to left
~	Bitwise complement	Right to left
+	Unary plus	Right to left
-	Unary minus	Right to left
++	Pre increment	Right to left
--	Pre decrement	Right to left
&	Address of	Right to left
*	Dereference	Right to left
(type)	Cast	Right to left
sizeof	Size in bytes	Right to left
new	Allocate memory	Right to left
delete	Deallocate memory	Right to left
.*	Direct pointer to class member selection	Left to right
->*	Indirect pointer to class member	Left to right

Operator	Description	Associativity
*	Multiplication	Left to right
/	Division	Left to right
%	Modulus	Left to right
+	Addition	Left to right
-	Subtraction	Left to right
<<	Bitwise shift left	Left to right
>>	Bitwise shift right	Left to right
<	Less than	Left to right
<=	Less than or equal to	Left to right
>	Greater than	Left to right
>=	Greater than or equal to	Left to right
==	Equal to	Left to right
!=	Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise inclusive OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional operator	Left to right
=, *=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=	Assignment operators	Right to left
,	Comma	Left to right

Elkövethető hibák

```
if ( i & mask == 0 )
```

vs.

```
if ( ( i & mask ) == 0 )
if ( i = 1 )
```

vs.

```
if ( i == 1 )
```

vs.

```
if ( 1 == i )
```

Operátorok

- C++: meglévő operátorok túlterhelése saját típusok esetén (nem mindegyik)
- Eiffel, Clean: újabb operátorok létrehozása
- Java: nem túlterhelhető operátorok
- C++11: `operator""`, user-defined literals

```
OutputType operator "" _suffix(long);
OutputType operator "" _suffix(long double);

OutputType evil = 666_suffix;
OutputType obj  = 1.5_suffix;
```

- Fizikai dimenziók támogatása típusrendszerrel
- Mars Climate Orbiter űrszonda

Kifejezések

- Típus (fordítási idő)
- Érték (jellemzően futási idő)
- Példák
  - `x + 500`
  - `x / 2`
  - `x = 3`
  - `x++`
  - `++x`
- Mellékhatás, eredmény

Bináris operátorok

- Mi az `a ⋆ b` kifejezés típusa?
- Bináris operátorok csak azonos típusú operandusokra vannak megírva.
- Mi van `a` és `b` típusa eltér? Mi történjen?

Szokásos aritmetikai konverzió

C++11 előtt

Ha a két operandus típusa eltér...

- Ha az egyik operandus `long double`, akkor másik is `long double`-lé konvertálódik.
- Különben, ha az egyik operandus `double`, akkor másik is `double`-lé konvertálódik.
- Különben, ha az egyik operandus `float`, akkor másik is `float`-tá konvertálódik.

C++11 előtt

Ha a két operandus típusa eltér...

- Különben, ha az egyik operandus `unsigned long`, akkor a másik is `unsigned long`-gá konvertálódik
- Különben, ha az egyik operandus `long int`, a másik egy `unsigned int`, akkor ha a `long int` típus képes ábrázolni az összes `unsigned int`-et, akkor az `unsigned int` `long int`-té konvertálódik, különben mindkét operandus `unsigned long int`-té konvertálódik.
- Különben, ha az egyik operandus `long`, akkor a másik is `long`-gá konvertálódik.
- Különben, ha az egyik operandus `unsigned`, akkor a másik is `unsigned`-dá konvertálódik.
- Különben, mindkét operandus `int`.

C++11

Ha a két operandus típusa eltér...

- Ha az egyik operandus `long double`, akkor másik is `long double`-lé konvertálódik.
- Különben, ha az egyik operandus `double`, akkor másik is `double`-lé konvertálódik.
- Különben, ha az egyik operandus `float`, akkor másik is `float`-tá konvertálódik.

C++11

Ha a két operandus típusa eltér...

- Különben, ha az egyik operandus `unsigned long long`, akkor a másik is `unsigned long long`-gá konvertálódik.
- Különben, ha az egyik operandus `long long int`, a másik `unsigned long int`, akkor, ha a `long long int` típus képes ábrázolni az összes `unsigned long int`-et, akkor az `unsigned long int long long int`-té konvertálódik, különben mindkét operandus `unsigned long long int`-té konvertálódik.
- Különben, ha az egyik operandus `unsigned long long`, akkor a másik is `unsigned long long`-gá konvertálódik.

C++11

Ha a két operandus típusa eltér...

- Különben, ha az egyik operandus `long int`, a másik egy `unsigned int`, akkor ha a `long int` típus képes ábrázolni az összes `unsigned int`-et, akkor az `unsigned int` `long int`-té konvertálódik, különben mindkét operandus `unsigned long int`-té konvertálódik.
- Különben, ha az egyik operandus `long`, akkor a másik is `long`-gá konvertálódik.
- Különben, ha az egyik operandus `unsigned`, akkor a másik is `unsigned`-dá konvertálódik.
- Különben, mindkét operandus `int`.

`bool` konverzió

- `true` $\rightarrow$ 1
- `false` $\rightarrow$ 0
- 0 $\rightarrow$ `false`
- *Egyébként* $\rightarrow$ `true`

Kiértékelési sorrend

```
bool f()
{
  std::cout << 'f';
  return false;
}

bool g()
{
  std::cout << 'g';
  return true;
}

bool h()
{
  std::cout << 'h';
  return false;
}
```

```
// ...
if ( f() == g() == h() )
{
  std::cout << 'i';
}
else
{
  std::cout << 'h';
}
```

Kiértékelési sorrend

- Hány karakter jelenik meg a kódrészlet lefuttatásakor?

Kiértékelési sorrend

- _ _ _ _

Kiértékelési sorrend

- `___i`

Kiértékelési sorrend

- f g h i

Kiértékelési sorrend

- g h f i

Kiértékelési sorrend

- h g f i

Kiértékelési sorrend

- f h g i

Kiértékelési sorrend

- h f g i

Kiértékelési sorrend

- g f h i

Kiértékelési sorrend

- Mi az `a ⋆ b` kifejezés kiértékelésének a sorrendje?
- Általában a kifejezéseken belüli részkifejezések kiértékelési sorrendje nem meghatározott.
- Nem tehető fel a balról jobbra kiértékelés
- Implementáció-függő
- Kódoptimalizációs eszköz
- Fordítóprogram nem figyelmeztet
- Hibás kódok alapja lehet
- Mellékhatások életbe lépése kérdéses

Példa

```
const int n = 10;
int v[ n ];
for( int i = 0; i < n; )
{
    v[ i++ ] = i;
}
for( int i = 0; i < n; )
{
    v[ i ] = i++;
}
```

Definiált kiértékelési sorrend

- `&&`
- `||`
- `,`

Definiált kiértékelési sorrend

```
bool pred(int x);

bool cond = ...;
if ( cond && pred( 3 ) )
{
}
if ( cond || pred( 1 ) )
{
}
```

Lusta vagy rövidzáras kiértékelés

Logikai kifejezések más nyelvekben

- Ada: default mohó kiértékelés, `and then`, `or else`
- Java: `&`, `&&`, `|`, `||`

Szekvenciapont

- ;
- Definiált kiértékelési sorrenddel rendelkező operátorok

Azonosítók a lefordított tárgykódban

r.cpp:

```
int f()
{
}

int f( int )
{
}

int f( double )
{
}
```

```
g++ -c r.cpp
```

```
nm r.o
```

```
000000000000000f T _Z1fd
0000000000000006 T _Z1fi
0000000000000000 T _Z1fv
```

Azonosítók a lefordított tárgykódban

r.cpp:

```
int f()
{
}

double f( void* )
{
}

namespace nspace
{
  int f( double )
  {
  }
}
```

```
g++ -c r.cpp
```

```
nm r.o
```

```
0000000000000006 T _Z1fPv
0000000000000000 T _Z1fv
0000000000000019 T _ZN6nspace1fEd
```

Azonosítók a lefordított tárgykódban

r.cpp:

```
namespace
{
  int f( double )
  {
  }
}

inline int f()
{
}

double f( void*,
          char* )
{
}
```

```
nm r.o
```

```
00...0011 T _Z1fPvPc
00...0000 t _ZN12_GLOBAL__N_11fEd
```

Azonosítók a lefordított tárgykódban

r.cpp:

```
class Class
{
public:
  int f(void*);
};

int Class::f(void*)
{
}
```

```
g++ -c r.cpp
```

```
nm r.o
```

```
00...000 T _ZN5Class1fEPv
```

Azonosítók a lefordított tárgykódban

r.cpp:

```
int x;

namespace A
{
  int x;
}

class Class
{
  static int c;
};

int Class::c;
```

```
g++ -c r.cpp
```

```
nm r.o
```

```
000...004 B _ZN1A1xE
000...008 B _ZN5Class1cE
000...000 B x
```