

# Típusok és kifejezések

Pataki Norbert



Programozási Nyelvek és  
Fordítóprogramok Tanszék

Programozási Nyelvek (C++)

# Lexikális egységek

- Kulcsszavak ✓
- Azonosítók ✓
- Konstans szövegliterálok ✓
- Konstansok: 1234, 12.34, stb.
- Operátorok: +, <<, ., stb.
- Szeparátorok: ;, , , , stb.

# Konstansok

	42	0x2A	052
Típus	int	int	int
Érték	42	42	42

Mi az, hogy int? Mi az, hogy típus? Mi az, hogy 42?

# Processzor

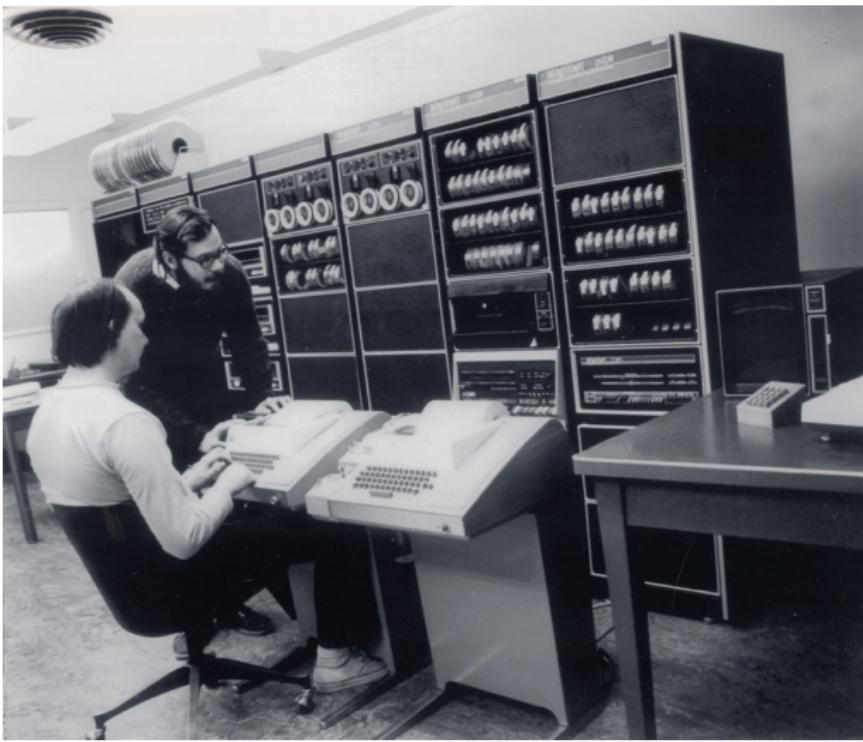
- Regiszterek, nem változók
- Típustalanság, méretek:
  - RAX, RBX, ... – 64 bit
  - EAX, EBX, ... – 32 bit
  - AX, BX, ... – 16 bit
  - AH, AL – 8 bit
  - Lebegőpontos regiszterek
  - Egyéb specifikus regiszterek



# Számítógépek története



## Számítógépek története



## Számítógépek története



# Számítógépek története



## Típus

- Bitsorozat: 11001001000100001011101
  - Jelentés megadása
  - Melyek az értelmes műveletek, mi az eredményük?
  - Fordítási visszajelzések
  - Optimalizációk
  - Absztrakciók

int

- int: alapértelmezett egésztípus
- Előjeles
- Kettes komplement

sizeof( int ): implementáció-függő (legyen optimális)

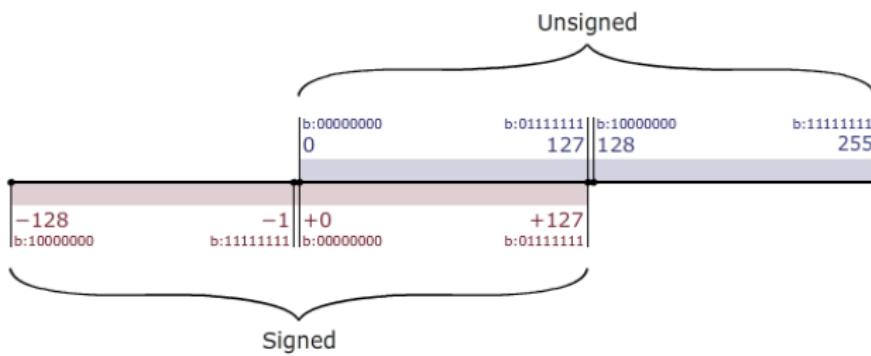
# Konstansok

	42L	42U	42UL
Típus	long	unsigned int	unsigned long
Érték	42	42	42

# Típusok mérete

- **char: a nyelv egységtípusa**
- `sizeof( char )`
- `sizeof( char ) ≤ sizeof( short )`
- `sizeof( short ) ≤ sizeof( int )`
- `sizeof( int ) ≤ sizeof( long )`
- `sizeof( bool ) ≤ sizeof( long )`

# Előjeles/előjel nélküli típusok



# Több egész szám ábrázolása

- `java.math.BigInteger`
- `Integer` (**Haskell**)

# Lebegőpontos konstansok típusa

	<b>12.34</b>	<b>12.34f</b>	<b>12.34L</b>
<b>Típus</b>	double	float	long double
<b>Érték</b>	12.34	12.34	12.34

# Lebegőpontos konstansok típusa

	12.34	88e-1	1234e-2f	54e-1L
Típus	double	double	float	long double
Érték	12.34	8.8	12.34	5.4

# Típusok mérete

- IEEE 754, lebegőpontos számábrázolás, következmények
- `sizeof( char )`
- `sizeof( float ) ≤ sizeof( double )`
- `sizeof( double ) ≤ sizeof( long double )`



# char

- char:
  - signed char
  - unsigned char

Implementáció-függő

# Operátorok jellemzése

- Precedencia
- Asszociativitás, kötés
- Aritás
- Fixitás
- Típus
- Mellékhatás, eredmény

# Precedencia

Precedence	Operator	Description	Associativity
1	::	Scope resolution	Left-to-right
2	a++ a-- type() type{} a() a[] . ->	Suffix/postfix increment and decrement Functional cast Function call Subscript Member access	
3	++a --a +a -a ! ~ (type) *a &a sizeof new new[] delete delete[]	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT C-style cast Indirection (dereference) Address-of Size-of <small>[note 1]</small> Dynamic memory allocation Dynamic memory deallocation	Right-to-left
4	* -*>	Pointer-to-member	Left-to-right
5	a*b a/b a%b	Multiplication, division, and remainder	
6	a+b a-b	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	<<>>	Three-way comparison operator ( <small>since C++20</small> )	
9	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
10	== !=	For relational operators = and ≠ respectively	
11	&	Bitwise AND	
12	^	Bitwise XOR (exclusive or)	
13		Bitwise OR (inclusive or)	
14	@@	Logical AND	
15		Logical OR	
16	a?b:c throw = += -= *= /= %= <<= >>= &= ^=  =	Ternary conditional <small>[note 2]</small> throw operator Direct assignment (provided by default for C++ classes) Compound assignment by sum and difference Compound assignment by product, quotient, and remainder Compound assignment by bitwise left shift and right shift Compound assignment by bitwise AND, XOR, and OR	Right-to-left
17	,	Comma	Left-to-right

# Példák

- 7 + x
- x++;
- x = 7 \* a + 5 \* b;
- x = y++ \* y++;
- x += ( y == 3 );
- 12 / 5
- 12. / 5

# Kiértékelés, példa

Mennyi az alábbi kifejezés értéke?

$$1 + 2 * 3 + 4$$

# Kiértékelés, példa

1 + 2 \* 3 + 4

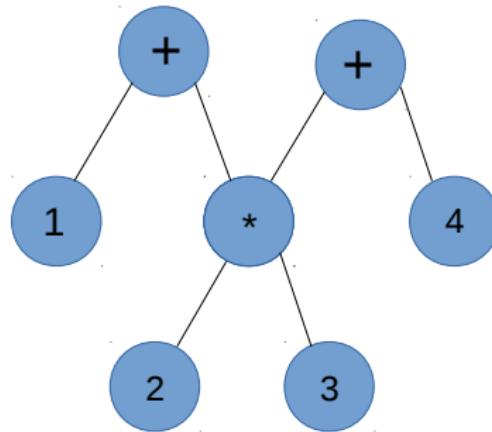
ugyanaz:

1 + ( 2 \* 3 ) + 4

# Kiértékelés, példa

11

# Absztrakt szintaxisfa(AST)



# Kötés

```
x = a + b + c + d; // balról
```

```
a = b = c = d = x; // jobbról
```

# Inkrementálás, dekrementálás

```
int i = 1;
```

```
std::cout << ++i; // 2
```

```
int i = 1;
```

```
std::cout << i++; // 1
```

# Inkrementálás, dekrementálás

- Eredmény
- Mellékhatás

//...

```
for( int i = 0; i < n; ++i )  
{  
    // ...  
}
```

# Értékadás – mit ír ki az alábbi kódrészlet?

```
int x = 2;  
  
if ( x = 0 )  
{  
    std::cout << 'A'  
        << std::endl;  
}  
  
if ( x = 4 )  
{  
    std::cout << 'B'  
        << std::endl;  
}
```

# Szekvenciapontok

- Biztosítja: a kifejezések ki vannak értékelve
- A mellékhatások végbementek
- Hiánya: portabilitási problémák
- Működési különbségek
- Optimalizációs lehetőség

# Példa

```
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 << ":-)";
}
else
{
    std::cout << ":-(";
}
```

# Szekvenciapontok

- ; – ;
- &&, || – működési jellegzetesség, lusta kiértékelés
- , – operátorként

# Példa

```
const int s = 5;  
int v[ s ];  
  
for( int i = 0; i < s; )  
{  
    v[ i ] = i++;  
}
```

# Szekvenciapontok

```
void f( int a )
{
    std::cout << a << ' '
                  << 1;
}
```

```
void f( int a, int b )
{
    std::cout << a << ' '
                  << b << ' '
                  << 2;
}
```

# Szekvenciapontok

```
void g()  
{  
    double d = (2, 5);  
  
    int x = 2;  
  
    f( x++, x++ );  
  
    f( ( x++, x++ ) );  
}
```

# Szekvenciapontok, kérdések

```
if ( f() && g() ) // ...
if ( f() || g() ) // ...
if ( f(), g() ) // ...
if ( g(), f() ) // ...
```