Operátory a jejich priority

1. (*podvýraz*) *ident* *literál* `_Generic`
2. [*index*] (*volání*) .*prvek* ->*prvek* `++` `--` (postfixové)
3. `++` `--` `sizeof` `_Alignof` `&` `*` `+` `-` `~` `!` (*typ*) (prefixové)
4. `*` `/` `%`
5. `+` `-`
6. `<<` `>>`
7. `<` `>` `<=` `>=`
8. `==` `!=`
9. `&`
10. `^`
11. `|`
12. `&&`
13. `||`
14. `?:`
15. `=` `+=` `*=` ... (asoc. zprava)
16. `,`

x + 1 > 5 && x < y + 2

(? :)

Priority: drobné zákeřnůstky

Chytáky:

```
(x&1)==(y&1)
```

```
1<<4 + 1<<5
```

Priority: drobné zákeřnůstky

Chytáky:

```
x&1 == y&1 ... x&(1==y)&1
```

```
1<<4 + 1<<5 ... 1<<(4+1)<<5
```

Konverze typů

Automatické konverze:

- Integery menší než `int` → `int` / `unsigned int`
- Pole → ukazatel na první prvek
- Funkce → ukazatel na funkci
- 0 ↔ null pointer (vyžaduje-li to kontext)
- Ke konverzím nedochází u `sizeof`, `_Alignof` a `&`

U binárních operátorů (mimo přiřazení a posuvů):

- `_Complex`
- `long double`
- `double`
- `float`
- “větší integer”
  - stejně velký unsigned a signed → unsigned

Konverze typů II

Na vyžádání (nebo při přiřazení):

- Integer na `bool`: 0 → 0, nenula → 1
- Integer na menší unsigned: modulo
- Integer na menší signed: závislé na implementaci (i trap)
- Float na integer: zaokrouhluje k nule (ořezává)
- Integer na float: může ztratit přesnost
- Complex na real: zahodí imaginární část

V parametrech funkce: `printf("%ld", (long) 42);`

- Typ je určen prototypem funkce, existuje-li. Jinak:
- Malé celočíselné typy se předají jako `int`
- `float` se předá jako `double`

Pořadí vyhodnocování

Liší se pořadí **vyhodnocování** (podle asociativity operátorů) a **provádění side-efektů** (vesměs nedefinováno).

Synchronizační body (sequence points):

- operátory `&&`, `||` a `?`
- čárka jako operátor
- konec výrazu (příkaz, podmínka v cyklu ...)
- volání funkce a návrat z ní

Mezi synchronizačními body je zakázáno měnit jeden objekt vícekrát nebo současně číst a zapisovat (výjimka: přiřazení).

Chyták: `printf("%d %d %d", a++, a++, a++);`

Deklarace

```
static volatile int *(* const f)[10] = NULL;
```

???

Deklarace

```
static volatile int *(* const f)[10] = NULL;
```

Třída uložení

- `static`, `extern`, `auto`, `register`, `_Thread_local`
- Ovlivňuje alokaci, dobu života i viditelnost
- Také sem patří `typedef`

Kvalifikátory

- `const`, `volatile`, `restrict`
- `_Alignas(...)`

Typy

- Čteme jako výraz

Inicializátor

- Pro staticky alokované implicitně 0

Deklarace

```
static volatile int *(* const f)[10] = NULL;
```

Třída uložení

- `static`, `extern`, `auto`, `register`, `_Thread_local`
- Ovlivňuje alokaci, dobu života i viditelnost
- Také sem patří `typedef`

Kvalifikátory

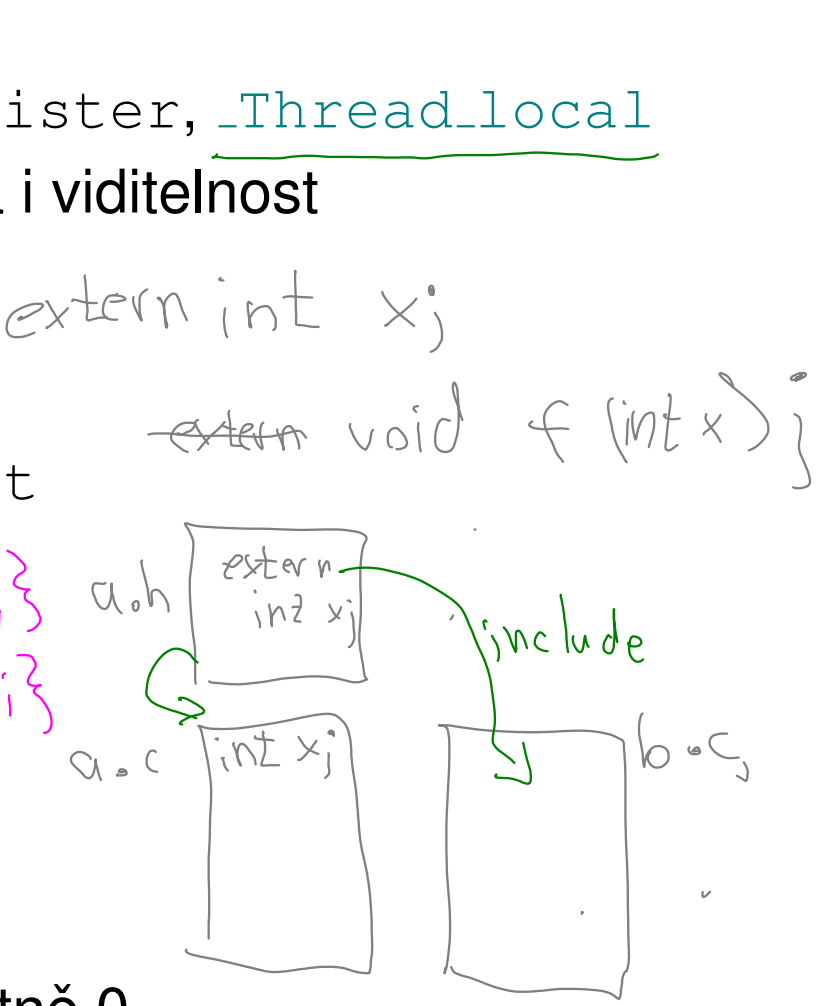
- `const`, `volatile`, `restrict`
- `_Alignas(...)`

Typy

- Čteme jako výraz

Inicializátor

- Pro staticky alokované implicitně 0



Deklarace funkcí

```
int main(int argc, char **argv);
```
(klasika)

```
int f();
```
(o argumentech nic neříkám)

```
int f(void);
```
(argumenty nemá)

```
int f(char x, float f);
```
(typované argumenty)

```
int f(int x, ...);
```
(jeden nebo více argumentů)

```
typedef struct { int x,y; } vec;
vec add(vec x, vec y);
```
(struktury předám hodnotou)

```
void sum(int pole[]);
```
(ale pole odkazem)

```
void sum(int n, int matice[n][n]);
int cmp(int x[restrict], int y[restrict]);

static inline int cmp(int x, int y);
```