BASIX — Reference příkazů

Stručná, systematická reference všech příkazů a konstrukcí jazyka BASIX.

Legenda:

`typ` = `int` / `real`

`[výraz]` = hranatými závorkami uzavřený výraz

`{ }` = volitelné

`...` = opakovatelné

Obsah

---

1. Proměnné

`var` — deklarace proměnné

```
var(typ, jmeno, [výraz])
var(typ, jmeno)
```

Prvek	Popis
`typ`	`int`, `real`, `(array, typ)`, `(obj, typ)`
`jmeno`	nový identifikátor
`[výraz]`	počáteční hodnota; **vynecháno** → výchozí nula (`0`/`0.0`)

- Zavádí novou proměnnou a rovnou ji inicializuje.
- Implicitní konverze typu výrazu na `typ` proměnné, pokud se liší.
- `(obj, typ)` nemá 3-argumentovou formu — instance se vždy inicializuje výchozími hodnotami členů (viz 8. `obj`).

Odkazy na varianty pro pole: 5.1, 5.6.

`set` — přiřazení hodnoty

```
set(jmeno, [výraz])                          ; skalár
set(jmeno[index], [výraz])                   ; prvek 1D pole
set(jmeno[i1][i2]...[iD], [výraz])           ; prvek D-rozměrného pole
set(jmeno.clen, [výraz])                     ; člen obj (lze řetězit .a.b.c)
set(pole, [výraz])                           ; A: skalár -> celé pole (text)
set(promenna, [pole])                        ; B: pole -> skalár (parsování)
set(pole1, [pole2])                          ; C: kopie hodnot mezi poli
set(cil, jmeno_funkce([v1], ...))            ; zachycení návratové hodnoty funkce
```

Prvek	Popis
`jmeno`	**již existující** proměnná (musí být předem `var`)
`[výraz]`	nová hodnota; implicitně konvertuje na typ cíle

- Nikdy nezavádí novou proměnnou a nemění statický typ cíle.
- U pole/indexu se index mimo meze saturuje (viz 5.4).
- Formy A/B/C viz 5.5 Zápis celého pole.

---

2. Výrazy a konverze

Obecný výraz

```
[výraz]
```

Gramatika `výraz`: literál | proměnná | operace (unární/binární) nad operandy | `jmeno[index]` | `jmeno.clen` | (jako výjimka na vyjmenovaných místech) holé `jmeno_pole`.

Explicitní konverze typu

```
[typ; výraz]
```

Vyhodnotí `výraz` ve vlastním typu, poté ho převede na `typ` (`real → int` = ořezání desetinné části, `int → real` = beze ztráty).

Implicitní konverze — tabulka

Z → Do	Pravidlo
`int → real`	přesně, beze ztráty
`real → int`	ořízne se desetinná část

Uplatní se automaticky při `var`, `set`, návratu z funkce, uložení výsledku operace jiného typu apod.

---

3. Bloky a řízení toku

`code` — pojmenovaný blok

```
code(label)
    tělo
end code
```

Provede `tělo` popořadě; neiterační (proběhne max. 1×).

`while` — cyklus s podmínkou na začátku

```
while({label;} [podmínka])
    tělo
end while
```

Test **před** každou iterací (i první). Nepravdivá napoprvé → tělo se neprovede vůbec.

`dowhile` — cyklus s podmínkou na konci

```
dowhile({label;} [podmínka])
    tělo
end dowhile
```

Test **po** každé iteraci → tělo proběhne vždy **alespoň jednou**.

`loop` — cyklus s krokováním

```
loop({label;} promenna, ([od], [do], {[krok]}))
    tělo
end loop
```

Prvek	Popis
`promenna`	**musí existovat** (dřívější `var`); typ libovolný
`od`, `do`	meze, vyhodnotí se **jednou**, na začátku; inkluzivní
`krok`	volitelný, výchozí `[1]`; vyhodnotí se **jednou**

- `krok >= 0` (ascendentní, včetně `0`) → test `promenna <= do`.
- `krok < 0` (descendentní) → test `promenna >= do`.
- Po těle: krokování `promenna = promenna + krok` (přes hierarchii typů, saturaci a implicitní konverzi).

`for` — cyklus přes pole

1D:

```
for({label;} promenna, pole)
    tělo
end for
```

Vícerozměrné (D proměnných = D rozměrů):

```
for({label;} (promenna1, ..., promennaD), pole)
    tělo
end for
```

Prvek	Popis
`promenna(X)`	musí existovat předem (`var`); typ libovolný
`pole`	existující pole s počtem rozměrů `D`

- Ekvivalent `loop(promenna, ([1], [velikost(pole)], [1]))`.
- U vícerozměrné varianty se poslední proměnná mění nejrychleji (row-major, „odometr”).
- Pole nikdy nemá 0 prvků → tělo proběhne vždy **alespoň jednou**.

`break` — předčasné ukončení

```
break(label)
break()
```

Forma	Cílí na	Platí uvnitř
`break(label)`	konkrétní obklopující blok libovolného ze 7 druhů (`code`, `while`, `loop`, `for`, `dowhile`, `if`, `case`)	kterýkoliv blok
`break()`	nejbližší obklopující **cyklus** (`while`/`loop`/`for`/`dowhile`)	jen uvnitř cyklu, jinak chyba kompilace

Ukončí cílový blok i všechny mezilehlé bloky mezi místem `break` a ním (víceúrovňové ukončení).

`continue` — přeskočení zbytku iterace

```
continue(label)
continue()
```

Cílí **jen na cyklus** (`while`/`loop`/`for`/`dowhile`) — nikdy na `code`, `if`, `case` (chyba kompilace).

Cyklus	Co se stane
`while`	rovnou vyhodnocení podmínky
`dowhile`	rovnou vyhodnocení podmínky
`loop`	nejdřív krokování (`+krok`), pak test
`for`	nejdřív krokování (`+1`, případně kaskádové u vícerozměrného), pak test

Kolize labelů (společné pravidlo pro `code`/`while`/`loop`/`for`/`dowhile`/`if`/`case`)

- Vnořený blok nesmí použít `label`, který už používá kterýkoliv z jeho (přímo/nepřímo) obklopujících bloků → chyba kompilace.
- Sourozenecké (nevnořené) bloky stejný `label` sdílet **mohou**.

---

4. Podmínky (`if`/`case`)

Čtyři varianty, odlišené druhým argumentem `if`.

4.1 Dvouhodnotová

```
if({label;} [výraz])
    case({label;} true)
        příkazy
    end case
    case({label;} false)
        příkazy
    end case
end if
```

- 1–2 `case` bloky, hodnoty `true`/`false` (bez duplicit).
- Pořadí zápisu case bloků nehraje roli — vybírá se podle hodnoty.
- Bez odpovídajícího `case` → no-op.

4.2 Řetězená — `first`

```
if({label;} first)
    case({label;} [výraz])   ; libovolný počet
        příkazy
    end case
    case({label;})            ; volitelný, NEJVÝŠE 1, musí být POSLEDNÍ
        příkazy
    end case
end if
```

- Provede se **první** case s pravdivou podmínkou, poté `if` ihned končí.
- Bez shody a bez catch-all case → no-op.

4.3 Vícenásobná — `every`

```
if({label;} every)
    case({label;} [výraz])   ; libovolný počet
        příkazy
    end case
    case({label;})            ; volitelný, NEJVÝŠE 1, musí být POSLEDNÍ
        příkazy
    end case
end if
```

- Provedou se **všechny** case s pravdivou podmínkou (bez short-circuit mezi nimi), catch-all case (je-li) se provede vždy naposled.
- `break(case_label)` uvnitř case ukončí **jen ten case**, `if` pokračuje dál dalším case.

4.4 Postupná — `dive`

```
if({label;} dive)
    case({label;} [výraz])   ; libovolný počet
        příkazy
    end case
    case({label;})            ; volitelný, NEJVÝŠE 1, musí být POSLEDNÍ
        příkazy
    end case
end if
```

- Prochází case bloky odshora, dokud vychází pravdivě — **první nepravdivá podmínka celý `if` okamžitě ukončí** (short-circuit).
- Catch-all case (dosažen jen po řadě samých pravdivých) se provede vždy.

4.5 Srovnávací tabulka

	dvouhodnotová	**`first`**	**`every`**	**`dive`**
max. `case`	2	∞ (+1 catch-all)	∞ (+1 catch-all)	∞ (+1 catch-all)
pořadí zápisu	nezáleží	rozhoduje (první výhra)	jen pořadí *provedení*	rozhoduje zásadně (řetěz)
“false” znamená	vybírá opačnou větev	jde dál	přeskočí, jde dál	**ukončí celý `if`**
`break(case_label)` = `break(if_label)`?	ano (vždy)	ano (vždy)	ne (obecně)	ne (obecně)

Ve všech čtyřech variantách: tělo `if` smí obsahovat výhradně `case(...) ... end case` bloky; `continue`/nepojmenovaný `break()` na `if`/`case` cílit nemohou.

---

5. Pole

5.1 Jednorozměrné pole — `var`

Varianta A (velikost jako výraz, výchozí nuly):

```
var((array, typ), jmeno, [velikost])
```

Výsledek < 1 → saturuje na `1`.

Varianta B (výčet hodnot, velikost = počet):

```
var((array, typ), jmeno, {položka1, položka2, ...})
```

Tvar položky	Přispívá
číselný literál `34`	1 hodnotou
řetězcový literál `"ab"`	1 hodnotou na znak (UTF-32 kód)
`[výraz]`	1 hodnotou
`[jmeno_jineho_pole]`	všemi jeho prvky (vícerozměrné = vynechá se)

Celkem 0 hodnot → saturuje na 1 prvek s výchozí nulou.

5.2 Čtení prvku

```
jmeno[index]
```

Je-li `index` typu `real` → implicitní konverze na `int` (ořezání).

5.3 Zápis prvku

```
set(jmeno[index], [výraz])
```

5.4 Saturace indexu

Podmínka	Použije se
`index < 1`	`1`
`index > velikost`	`velikost`
jinak	beze změny

5.5 Zápis celého pole (`set`)

```
set(pole, [výraz])       ; A: skalár -> pole, jako text (viz print/input)
set(promenna, [pole])    ; B: pole -> skalár, parsuje se jako input
set(pole1, [pole2])      ; C: kopie hodnot, prvek po prvku, s int/real konverzí
```

Kratší zdroj → cíl se doplní nulami; delší zdroj → přebytek se zahodí; velikost cíle se nikdy nemění.

5.6 Vícerozměrné pole — `var`

```
var((array, typ), jmeno, ([vyraz1], [vyraz2], ..., [vyrazD]))
```

- `D` = počet `[vyrazX]` v seznamu (`D=1` je totožné s 1D variantou A).
- Každý rozměr saturuje nezávisle na `1`, je-li menší.
- Žádná obdoba `{}` výčtu hodnot pro vícerozměrné pole neexistuje.

5.7 Čtení/zápis prvku vícerozměrného pole

```
jmeno[v1][v2]...[vD]                    ; čtení
set(jmeno[v1][v2]...[vD], [výraz])      ; zápis
```

Počet indexů musí přesně odpovídat `D` (žádná částečná indexace).

5.8 Saturace indexu (vícerozměrné)

Stejné pravidlo jako 5.4, aplikované **nezávisle v každém rozměru**.

5.9 Porovnání polí

```
[a == b]
[a != b]
```

Porovnává **jen velikost posledního rozměru** (obsah se ignoruje); funguje pro libovolnou kombinaci 1D/vícerozměrných polí a typů prvků. Výsledek je vždy `int 0`/`1`.

5.10 Implicitní konverze mezi poli / poli a skalárem

Řídí obecný mechanismus uplatňovaný v `set` (5.5) a v `{[jmeno_pole], ...}` (5.1B, tvar 4):

1. Vícerozměrné pole se linearizuje **row-major** (poslední rozměr nejrychlejší).
2. Prvek po prvku implicitní `int ↔ real`.
3. Nesoulad délky → doplnění nulou / zahození přebytku.

Netýká se `obj` (viz 8.5).

---

6. Vstup/výstup

`print`

```
print([výraz])                      ; A: skalár, nebo holé jméno pole (jako text)
print({položka1, položka2, ...})    ; B: série položek (4 tvary jako u 5.1B)
print()                             ; C: bez argumentu
```

Situace	Chování
skalár (`[výraz]`, tvar 3)	vypíše se jako desítkové číslo
holé jméno pole (`[jmeno_pole]`, tvar 4)	vypíše se jako **text** (Unicode znaky)
holé jméno **vícerozměrného** pole	tiše se **vynechá** (nic se nevypíše)
číselný literál (tvar 1)	vypíše se jako číslo
řetězcový literál (tvar 2)	vypíše se jako text
`real` hodnota	*shortest round-trip* zápis, vždy s `.0`/desetinnou tečkou
`print()` / `print({})`	vypíše **jeden konec řádku**

Žádné implicitní mezery/oddělovače/odřádkování mezi položkami (kromě výslovně prázdného volání).

`input`

```
input(promenna)
```

Cíl `promenna`	Chování
skalár (`int`/`real`)	přeskočí bílé znaky, přečte nejvýš 1 hodnotu z řádku; zbytek řádku se zahodí; neplatné/EOF → `0`/`0.0`; mimo `[-2^53, 2^53]` → saturace
jednorozměrné pole	uloží celý přečtený řádek znak po znaku, max. do velikosti pole; kratší text → doplní nulou; delší → zahodí přebytek

Vždy se spotřebuje **celý jeden řádek** vstupu, bez ohledu na to, kolik z něj bylo skutečně využito.

---

7. `max` / `min`

Zápis	Vrací
`max([výraz])`	horní mez `2^53` **typu** výrazu (hodnota výrazu se nepoužije)
`min([výraz])`	dolní mez `-2^53` typu výrazu
`max(jmeno)` (pole)	nejvyšší platný index posledního rozměru (= délka u 1D)
`min(jmeno)` (pole)	vždy `1`
`max((jmeno, [vyraz]))`	nejvyšší index rozměru číslo `vyraz` (číslováno od 1)
`min((jmeno, [vyraz]))`	vždy `1`
`max(jmeno)` (`obj` instance)	počet členů typu
`min(jmeno)` (`obj` instance)	vždy `1`

- Pro pole i `obj` vrací vždy `int`.
- `vyraz` u třetí podoby mimo `[1, D]` → saturuje se.
- Dvouargumentová `max(a, b)` (výběr většího ze dvou čísel) **neexistuje**.

---

8. `obj`

8.1 Definice typu

```
obj(jmeno_typu)
    var(typ, jmeno_clenu, [výraz])   ; libovolně (i)
    var(typ, jmeno_clenu2)           ; opakovaně, aspoň 1×
end obj
```

`typ` člena může být `int`, `real`, pole (1D/vícerozměrné), nebo jiný, dříve definovaný `obj`. Minimálně 1 člen, žádná horní mez.

8.2 Instance

```
var((obj, typ), jmeno)
```

Žádný 3. argument — vždy se naplní výchozími hodnotami členů (viz 8.4).

8.3 Čtení/zápis členu

```
jmeno_instance.jmeno_clenu                    ; čtení, lze řetězit a.b.c
set(jmeno_instance.jmeno_clenu, [výraz])      ; zápis
```

8.4 Výchozí hodnota členu

Výraz u definice člena se vyhodnocuje **znovu při každé instanci**, ne jednou staticky při definici `obj`.

8.5 Konverze `obj` ↔ jiné typy

Neexistuje žádná (ani implicitní, ani explicitní) konverze mezi `obj` a skalárem, jiným `obj`, nebo polem. Tam, kde by k ní mělo dojít, se uplatní **tiché vynechání** (hodnota zůstane beze změny, není to chyba kompilace).

8.6 Jmenný prostor

Jména `obj` typů tvoří **vlastní, oddělený** jmenný prostor od proměnných, polí, labelů i funkcí — nekolidují s nimi. Dva `obj` typy stejného jména ve stejném rozsahu jsou chyba kompilace.

---

9. Funkce

9.1 Definice

```
fun(jmeno)
    ret(typ, jmeno_ret, {[výraz]})
    par(typ, jmeno_par, {[výraz]})    ; libovolně mnoho, i 0
    ...
    tělo
end fun
```

Prvek	Pravidlo
`ret(...)`	**povinný**, vždy **první**, přesně jeden
`par(...)`	volitelné, libovolný počet, vždy až za `ret`
`typ` u `ret`/`par`	`int`, `real`, `(array, typ)`, `(obj, typ)`

9.2 `return`

```
return
```

Holé klíčové slovo bez argumentu. Okamžitě ukončí funkci (bez ohledu na zanoření); vrácená hodnota = aktuální `ret` v daném okamžiku. Přirozené doběhnutí `tělo` na `end fun` je rovnocenná (implicitní) forma návratu.

9.3 Volání

```
jmeno([vyraz_ret], [vyraz_par1], [vyraz_par2], ...)
```

- Vždy **samostatný příkaz** (nikdy uvnitř `[výraz]`).
    1. argument → `ret`, 2. → 1. `par`, 3. → 2. `par`, atd. (poziční).
- Chybějící argumenty → výchozí hodnota daného parametru.
- Argumenty se vyhodnocují **zleva doprava**, teprve poté se dosadí.

9.4 Zachycení návratové hodnoty

```
set(cil, jmeno([vyraz_ret], ...))
```

`cil` musí typem odpovídat `ret`. Nejvíc smysl dává pro `ret` typu `int`/`real`.

9.5 Předávání parametrů

Typ parametru	Způsob	Poznámka
`int`, `real`	**hodnotou**	kopie; změny uvnitř se ven nepropíšou
`(array, typ)`, `(obj, typ)`	**odkazem**	stejná instance jako u volajícího; zápis se ven promítne

Pro argument typu pole/`obj` se místo `[výraz]` píše `[jmeno]` — **holé jméno** existující proměnné (nebo jiného `par`/`ret`) **přesně** odpovídajícího typu, bez konverze.

9.6 Jmenný prostor a kolize

Jména funkcí tvoří vlastní jmenný prostor (nekolidují s proměnnými, poli, labely, `obj` typy). Dvě

funkce stejného jména kdekoliv v programu = chyba kompilace (funkce žijí na jediné globální úrovni).

9.7 Globální úroveň a pořadí

- Funkce se smí definovat **jen** na nejvyšší úrovni programu (nikdy vnořeně do bloku ani do jiné funkce).
- Na pořadí zápisu (a volání) funkcí **nezáleží** → dopředná volání, přímá i nepřímá (vzájemná) rekurze jsou vždy platné.
- Každé (i rekurzivní) volání má vlastní, nezávislou instanci `ret`/`par`.

---

10. Operátory — tabulka priorit

Priorita	Operátory	Asociativita
1 (nejvyšší)	unární `-`, `not`	zprava doleva
2	`* / %`	zleva doprava
3	binární `+ -`	zleva doprava
4	`< <= > >=`	zleva doprava
5	`== !=`	zleva doprava
6	`and`	zleva doprava
7 (nejnižší)	`or`	zleva doprava

- Žádné bitové operátory (`& | ^ ~ << >>`).
- `and`/`or` — short-circuit, výsledek vždy `int` `0`/`1`.
- Smíšený `int`/`real` operand → `int` se promuje na `real` (`real > int`); u aritmetiky se výsledek saturuje, u porovnání ne.

---

11. Klíčová slova

Kategorie	Slova
s vlastní (	`var` `set` `code` `while` `loop` `for` `dowhile` `if` `case` `break` `continue` `obj` `max` `min` `input` `print` `fun` `ret` `par`
holá (bez ())	`end` `return` `true` `false`
typová	`int` `real`

Identifikátor (proměnná, pole, label, `obj` typ, funkce) nesmí být shodný s žádným z výše uvedených klíčových slov. `first`/`every`/`dive` klíčovými slovy **nejsou** (vyskytují se jen na jednom místě, žádná kolize nehrozí).