Principy vyšších programovacích jazyků

Jitka Kreslíková, Aleš Smrčka

2021

Fakulta informačních technologií
Vysoké učení technické v Brně

IZP – Základy programování

Principy vyšších programovacích jazyků

- Ukládání informací
- Zpracování informací

Ukládání informací

- Reprezentace údajů.
- Datové struktury.
- Datové typy.
- Prvky programovacích jazyků.
- Proměnné, konstanty.

Reprezentace údajů

- Všechna data na počítači lze rozložit na jednotlivé bity,
- psát programy zpracovávající výlučně bity je nepohodlné,
- zavedení **datového typu** umožní určit, jak se budou konkrétní sady bitů používat,
  - různé pohledy na stejná data
- operátory (funkce) – umožní určit operace, jež budeme nad daty provádět.

Datové struktury

Datová struktura je:

- **homogenní** – všechny komponenty dané struktury jsou téhož typu.

  Příklad: Pole záznamů (i když záznam sám o sobě je heterogenní)

- **heterogenní** – komponenty struktury nejsou téhož typu.

  Příklad: Záznam

- **statická** – nemůže měnit v průběhu výpočtu počet svých komponent ani způsob uspořádání.
- **dynamická** – může měnit v průběhu výpočtu počet svých komponent a způsob uspořádání struktury.

  Příklad: uspořádaný binární strom lze transformovat např. na oboustranně vázaný lineární seznam a naopak.

- homogennost/heterogennost, statičnost/dynamičnost se posuzuje vždy jako vlastnost na jisté úrovni abstrakce.

Datové typy

- Návrh datových struktur je v programovacích jazycích podporován koncepcí datových typů a tzv. typovou kontrolou.
- **Datový typ** = množina hodnot a množina operací nad těmito hodnotami.
- Každý objekt (konstanta, proměnná, výraz, funkce) přísluší právě k jednomu datovému typu.
- Typ proměnných se zavádí v deklaraci proměnné.
- Typ výrazu je dán použitými operátory a operandy.
- Příslušnost k typu
  - syntaktická vlastnost objektu
  - pravidla jazyka určují, kdy a kde lze používat objekty kterých typů.

Datové typy

- Když zapisujeme operaci, musíme se ujistit, že její operandy i výsledek jsou správného typu.
- V některých situacích se provádí implicitní konverze typu (nutno se seznámit s pravidly pro implicitní konverze).
- Jindy používáme přetypování neboli explicitní konverzi.

explicitní konverze

Příklad: x a n jsou celá čísla, výraz: ((float)x) / n

implicitní konverze

Datové typy

- V dnešní době uvažujeme o datových typech více v souvislosti s potřebami programů, než s možnostmi stroje.
- Důraz se klade na přenositelnost programů.

Příklad:
Uvažujeme o **short int** jako o datovém objektu, jenž může pojmout hodnoty mezi -32768 a 32767, a ne jako o šestnáctibitovém objektu. Navíc koncept celého čísla zahrnuje operace, jež nad ním provádíme: součet, násobení atd. Rozsah hodnot, Prefix, [on line, cit. 2015-09-25]

Datové typy – vlastnosti

Datový typ je určen:

- názvem (logical, Boolean, int, real, float, den, pohlaví),
- množinou hodnot, kterých mohou nabývat konstanty, proměnné, výrazy, funkce určitého typu,

  Příklad: true, false, 358, 2.5, "Ahoj", sobota, 'a'

  - počet různých hodnot příslušejících typu T se nazývá **kardinalita** typu T.

Datové typy - vlastnosti

- množinou operací (každý operátor nebo funkce předpokládá operandy stanoveného typu a dává výsledek stanoveného typu.

Jestliže operátor připouští operandy několika typů (např. + pro sčítání jak čísel reálných tak čísel celých), pak se typ operace + určí podle dalších pravidel jazyka.

Příklad:
int / int … celočíselné dělení – výsledkem je int
int / float … dělení v pohyblivé řádové čárce – výsledkem je float
float / int … dělení v pohyblivé řádové čárce – výsledkem je float
float / float … dělení v pohyblivé řádové čárce – výsledkem je float

Datové typy - vlastnosti

Tři úrovně dostupných operací:

- operace se standardními datovými typy (aritmetické operace, logické operace),

Příklad operací: %, and, or, not, eq, xor, +, /, *, in

- operace dostupné v knihovnách funkcí (např. funkce pro práci s lineárními seznamy, vektory, frontou, zásobníkem, atd.),

Příklad operací: listInit, copyFirst, deleteFirst

- operace definované uživatelem (programátorem).

Příklad operací: vycisliPolynom

Hornerovo schéma

Datové typy - vlastnosti

- množinou atributů
  - Určují, které vlastnosti objektů daného typu jsou programově dosažitelné.

Příklad :
FIRST, LAST - pro datový typ integer v Adě.

Deklarujeme-li v Adě proměnnou A: integer; pak nejvyšší zobrazitelné celé číslo přiřadíme do této proměnné použitím atributu LAST :

```
A := integer LAST;
```

Datové typy - vlastnosti

symbolické konstanty

Příklad :
V jazyku C pro celočíselné typy: <limits.h>

CHAR_BIT	**SCHAR_MIN**	**SCHAR_MAX**	**UCHAR_MAX**
CHAR_MIN	**CHAR_MAX**	**MB_LEN_MAX**	**SHRT_MIN**
SHRT_MAX	**USHRT_MAX**	**INT_MIN**	**INT_MAX**
UINT_MAX	**LONG_MIN**	**LONG_MAX**	**ULONG_MAX**
LLONG_MIN	**LLONG_MAX**	**ULLONG_MAX**	

int a = INT_MAX – přiřazení maximální dosažitelné hodnoty do proměnné typu int (v daném prostředí).

V jazyku C pro reálné typy: <float.h>

Úkol: zjistěte ve vašem vývojovém prostředí všechny maximální dosažitelné hodnoty celočíselných a racionálních typů.

Datové typy – definování

- Nový datový typ – obvykle pomocí dříve definovaných datových typů.
- Hodnoty takových typů – složeny z hodnot komponent dříve definovaných ustavujících (konstitučních) typů a proto těmto složeným hodnotám říkáme **strukturované**.
- Jsou-li všechny hodnoty téhož konstitučního typu, pak tomuto typu říkáme **bázový typ**.

Datové typy – definování

- Konstituční typy mohou být opět samy strukturovány a tak lze vytvářet hierarchicky uspořádané struktury.
- Komponenty na nejnižší úrovni však jsou již atomické – dále nedělitelné. Proto je nutné, aby byly zavedeny také primitivní (jednoduché), nestrukturované typy.
- Hodnoty primitivních typů se mohou stanovit vyjmenováním (výčtem, enumerací).
- Mezi primitivními typy jsou tzv. standardní typy, které jsou předdefinovány.
  - Obvykle zahrnují čísla, logické hodnoty a znaky.

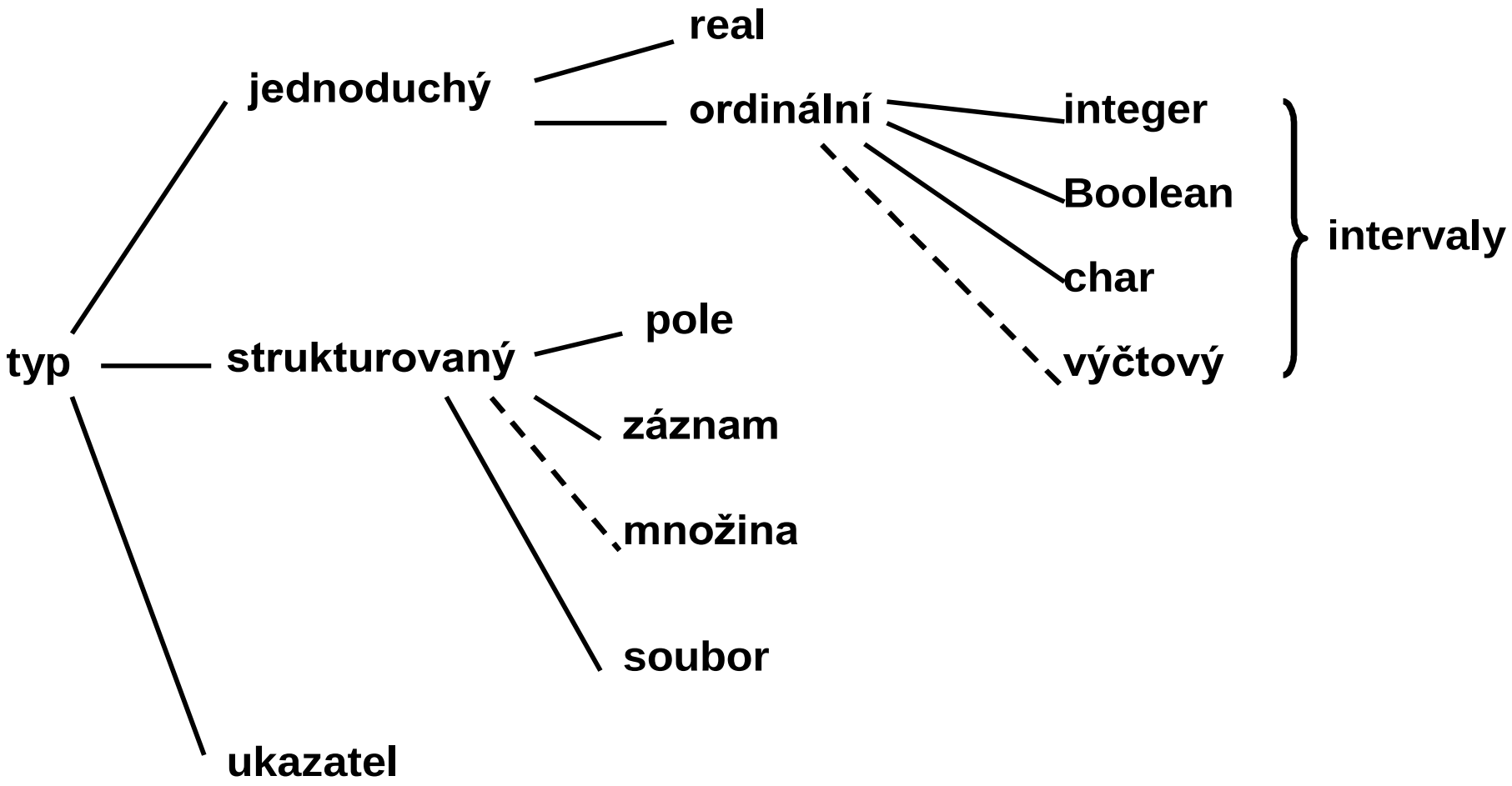
Datové typy – definování

- V případě deklarace vyjmenováním jsou jejich hodnoty seřazeny podle pořadí v posloupnosti vyjmenování.
- S těmito nástroji lze definovat primitivní typy a vytvářet složité datové struktury do libovolné vnořené úrovně.
- Existujte několik metod strukturování
  - Liší se operátory pro konstrukci hodnoty struktur a operátory pro výběr komponenty ze strukturované hodnoty.
  - Je to nutné kvůli praktickým problémům reprezentace dat a jejich využití.
- Mezi základní metody strukturování patří:
  - pole,
  - záznam,
  - množina,
  - soubor.

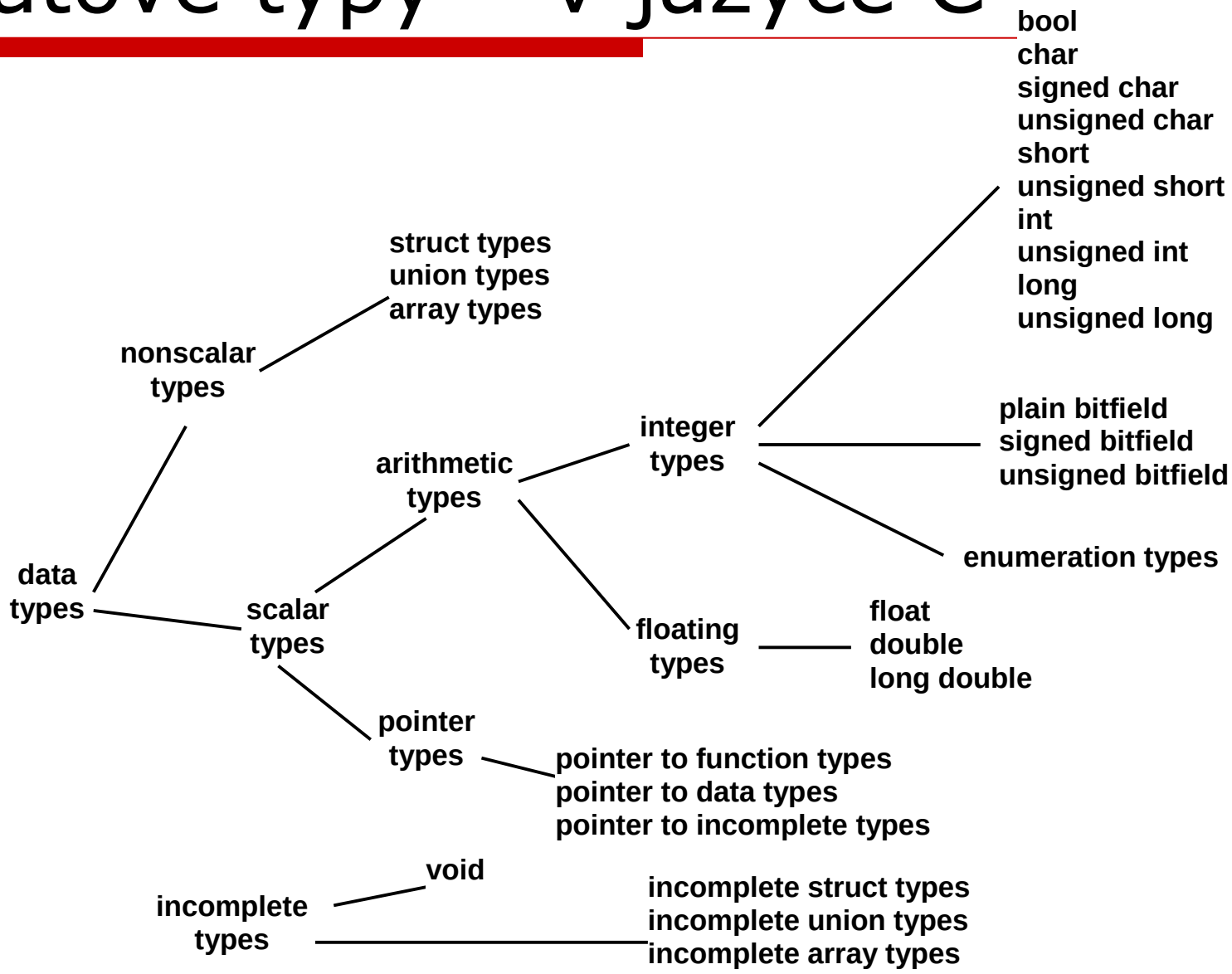
Datové typy – operátory

- **Ekvivalence** – bývá definována nad primitivními, resp. zabudovanými typy.
- **Přiřazení** – operace přiřazení je definována obvykle nad všemi datovými typy.
  - Pro rozsáhlé a hluboce strukturované typy však může taková operace představovat značné množství strojových instrukcí.
  - U strukturovaných typů může být realizace přiřazení realizována různým způsobem (např. kopie struktury po jednotlivých složkách).
- **Transformační operátory** – mapují jeden datový typ do jiného (přetypování).
- Strukturované hodnoty se vytvářejí ze svých komponent pomocí tzv. **konstruktorů**.
- Hodnotu jedné komponenty lze ze strukturované hodnoty získat pomocí **selektoru**.

Datové typy – rozdělení (Pascal)



Datové typy – v jazyce C



Datové typy – v jazyce C

Modifikátory typu:

- **long** – int, double
- **short** – int
- **signed** – znaménkový
- **unsigned** – neznaménkový

bit číslo	7	6	5	4	3	2	1	0	
MSB	1	0	1	1	0	0	1	1	LSB

Rozdíl mezi signed a unsigned je v rozsahu čísla:

unsigned: od 0 do $2^n - 1$,

pro char: 0 až 255

signed: od -2^{n-1} do $2^{n-1} - 1$

pro signed char: -128 až +127

Datové typy - v jazyce C

- Standard C zaručuje, že jednotlivé typy z hlediska alokace paměti jsou ve vztahu:

sizeof(char) = 1 Byte

sizeof(short int) ≤ sizeof(int) ≤ sizeof(long int)

sizeof(unsigned int) = sizeof(signed int)

sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)

Datové typy - paměťové nároky

Informativní přehled datových typů v jazyce C, spolu s možnými paměťovými nároky a jejich významem.

datový typ	**počet bitů**	**význam**
char, unsigned char, signed char	8	znak, malé celé číslo
short, unsigned short, signed short	16	krátké celé číslo
int, unsigned int, signed int	16 nebo 32	celé číslo
long, unsigned long, signed long, long long	32|64	dlouhé celé číslo
enum	8|16|32	výčtový typ
float	32	racionální číslo
double	64	racionální číslo s dvojitou přesností
long double	80|96	dlouhé racionální číslo
pointer	16|32|64	ukazatel

Jednoduché datové typy

- čísla
- znaky
- logické typy (Booleovský typ - bool)
- (ukazatele)

Mají souvislost s možnostmi stroje:

- velikost registrů
- organizace čísel na paměťovém médiu

Operace:

- přiřazení
- relace ekvivalence a uspořádání

Standardní datové typy

- **Typ int (integer)**
  - Reprezentuje množinu celých čísel, která je v daném programovacím jazyce k dispozici jako **standardní datový typ**.
  - Na rozdíl od matematického chápání množiny celých čísel jako množiny, která je nekonečná a je podmnožinou reálných čísel, datový typ int **není** nekonečná množina.
  - Konečnost kterékoliv množiny čísel reprezentované v počítači je nutným důsledkem konečnosti jeho paměti.

aritmetické operátory: +, -, *, /, modulo

relační operátory: =, ≠, <, ≤, ≥ , >

Standardní datové typy

- **Typ float (real)**
  - Skutečnost, že počítače pracují vždy s konečnými množinami hodnot má silný dopad na reprezentaci reálných čísel.
  - Výsledky s operandy typu float jsou vždy pouze přibližné, aproximující přesné hodnoty (na rozdíl od int).
  - Příčinou je vlastnost množiny reálných čísel (kontinua) – libovolně malý interval na reálné ose obsahuje nekonečně mnoho reálných čísel.
  - V programování typ float proto **nereprezentuje nekonečnou** a **nespočetnou** množinu reálných čísel, ale konečnou množinu reprezentativních intervalů reálného kontinua – konečnou podmnožinu racionálních čísel.

Příklad :1.345 reprezentuje interval <1.345, 1.346)
1035. reprezentuje interval <1035.,1036.)

Standardní datové typy

- "Nebezpečnými" operacemi aritmetiky čísel typu float jsou operace odčítání a dělení. U všech aritmetických operací je nutné také počítat s chybou zaokrouhlení.
- Jsou-li odečítána dvě velmi blízká malá čísla, pak se ve výsledku mohou ztratit téměř nebo úplně.

Příklad : $2e^{-40} - 1e^{-40} \rightarrow 0.000000$

- Podobně je-li dělitel velmi malý, pak snadno dojde k přeplnění výsledku dělení.

Nevhodný výraz : $abs(t/x) \leq eps$
je nutné použít: $abs(t) \leq eps \times abs(x)$

Standardní datové typy

Typ char

Existují standardizované soubory znaků:

- Kódy ISO (International Organisation for Standardisation)

  Mezinárodní standard ISO/IEC 10646:2017 definuje Universal Character Set (UCS). [on line, cit. 2019-09-25]

- ASCII (American Standard Code for Information Interchange), [on line, cit. 2019-09-25]
  tabulka ASCII: http://www.labo.cz/mft/matasciit.htm,

- EBCDIC (Extended Binary Coded Decimal Interchange). [on line, cit. 2019-09-25]

Standardní datové typy

- Typ char
  - Unicode - verze 14.0 obsahuje ~144 697 znaků světových abeced. Poslední verze Unicode 14.0.0

    [on line, cit. 2021-09-14]
  - UTF-8 je zkratka pro UCS Transformation Format.

    [on line, cit. 2018-09-24]
  - Proč Unicode?: http://www.cestina.cz/kodovani/unicode.html

    [on line, cit. 2019-09-25]

Standardní datové typy

- **Typ bool (Boolean)**
  - hodnoty typu bool = (false,true),
  - operace následník(false) = true,
    předchůdce(true) = false,
  - pořadí(false) = 0, pořadí(true) = 1

operátory: && (AND), || (OR), ! (NOT)

Výsledek typu bool dávají také všechny relační operátory (==, !=, <, ≤, ≥ , >).

(Pozor na C – tam je výsledkem relačních operátorů typ int!).
#include <stdbool.h>

Standardní datové typy

- **Typ interval**
  - jen v některých programovacích jazycích
  - lze definovat pouze nad ordinálními typy

  Příklad: type T = min .. max

  type Rok = 1900 .. 1999

Využití redundantní informace skýtané typem interval pro detekci chyb je jedním ze základních vlastností některých vyšších programovacích jazyků.

Standardní datové typy

- **Typ definovaný výčtem**
  - jednoduchý, ordinální typ

type den = $(C_1, C_2, \ldots C_n)$,

kde C_i je vždy identifikátor i-té hodnoty typu, n je kardinalita typu.

Objektem C_i nikdy nemůže být číslo, znak či textový řetězec.

Příklad :

type den = (pondeli,utery,streda,ctvrtek,patek,sobota,nedele)

Operace pořadí (ord): $\{ i_1, i_2, \ldots i_n \} \rightarrow \{ 0,1, \ldots , n-1 \}$

$ord(i_k) = k-1$, pro $k = 1, \ldots n$,

Poznámka: v některých programovacích jazycích lze vnutit vlastní pořadí (C).

Typový systém programovacího jazyka

Soubor pravidel, která přiřazují výrazům v daném jazyce typ.

Příklad :
Máme-li například výraz 2 + 3, typový systém přidělí tomuto výrazu nejspíše typ `int`.
Výrazu 2 * (3.14 - 2) přidělí nejspíše typ `double`.
Pro výraz 7 + "ABCD" však typový systém typ nenajde.

- Říkáme proto, že typový systém akceptuje výraz, pokud pro něj nalezne typ a zamítne výraz, pokud k němu typ nenalezne.
- Výrazy zamítnuté typovým systémem představují chybu a nemohou být provedeny.

Typový systém programovacího jazyka

- Samotný proces přiřazování typu výrazům nazýváme též **typovou kontrolou** výrazů.
- Rozeznáváme dva druhy typových systémů:
  - typový systém silný – akceptuje pouze bezpečné výrazy
  - typový systém slabý

Bezpečné výrazy jsou přitom takové, které nemohou za běhu programu způsobit chybu.

Typový systém, který není silný se nazývá slabý.

Typový systém programovacího jazyka

- Slabý typový systém může dovolit provedení výrazů, které nejsou bezpečné.
  - Hrozí tak nouzové zastavení programu chybou za běhu výpočtu (nebo ještě hůře – počítání s chybnou hodnotou bez jakékoli výstrahy).
- Silný typový systém tento problém nemá.
  - Na druhé straně zase zakazuje i provedení některých výrazů, které jsou bezpečné.
- Extrémním případem silného typového systému by byl systém, zamítající všechny výrazy.

Prvky programovacích jazyků

Znaky, které tvoří program, se spojují do **lexikálních atomů**.
Překladač vždy sestavuje z přicházejících znaků zleva doprava nejdelší možný atom, i když výsledek nemusí tvořit platný program v jazyku C:

Příklad :

Znaky jazyka	**Atomy**
forwhile	forwhile
b>x	b,>,x
b->x	b,->,x
b--x	b,--,x
b---x	b,--,-,x

Prvky programovacích jazyků

Existuje 5 tříd atomů:

- rezervované slovo
- identifikátor
- konstanta
- operátor
- oddělovač

Prvky programovacích jazyků

Oddělování sousedících atomů:

Příklad :
(a + b, a+b) - libovolný počet "bílých znaků"

do if, doif - minimálně jeden "bílý znak"

- Bílý znak
  - mezera, tabelátor,
  - nový řádek, posun řádku (LF),
  - návrat vozíku (CR), nová stránka, vertikální tabelátor
  - komentář

Prvky programovacích jazyků

Použitá notace:
Syntaktické kategorie (nonterminály) jsou indikovány *kurzívou.*
Základní symboly (terminály) jsou označeny **tučně.**
Dvojtečka (:), která následuje nonterminál uvádí jeho definici.
Alternativní definice jsou uvedeny na novém řádku, některým předchází slova "one of".
Volitelný symbol je indikován "opt".

Příklad : **{** *expression* $_{opt}$ **}** znamená volitelný výraz (expression) uzavřený ve složených závorkách.

Poznámka:
V následujícím textu budou prvky programovacích jazyků demonstrovány na gramatice jazyka C. Pokud nepůjde o konstrukci jazyka C, bude to zvlášť zdůrazněno.

Prvky programovacích jazyků

Lexikální elementy:

token:
- *keyword*
- *identifier*
- *constant*
- *string-literal*
- *punctuator*

Rezervovaná slova

- jsou vyhrazené identifikátory
- nelze vytvořit stejný identifikátor jako je některé rezervované slovo
- elementy definované přímo jako součást gramatiky jazyka
- uživatel nemá možnost je měnit – jsou zakomponována přímo do překladače
- různé implementace překladače mohou přidávat nová rezervovaná slova

Rezervovaná slova

keyword: one of

auto	**enum**	**restrict**	**unsigned**
break	**extern**	**return**	**void**
case	**float**	**short**	**volatile**
char	**for**	**signed**	**while**
const	**goto**	**sizeof**	**_Bool**
continue	**if**	**static**	**_Complex**
default	**inline**	**struct**	**_Imaginary**
do	**int**	**switch**	
double	**long**	**typedef**	
else	**register**	**union**	

Identifikátory

Identifikátor

- je jednoznačný název (pojmenování) používaný v programovacím jazyce k označení podprogramů, proměnných, datových typů, konstant, atd.
- může být libovolně dlouhý, ale dle konkrétní implementace se rozlišuje pouze prvních *n* (např. 31) znaků (ISO C99 – 63 znaků),
- musí být souvislý. Nelze jej přerušovat bílými znaky.

Identifikátory

[C99]

identifier:
identifier-nondigit
identifier identifier-nondigit
identifier digit

digit: one of
0 1 2 3 4 5 6 7 8 9

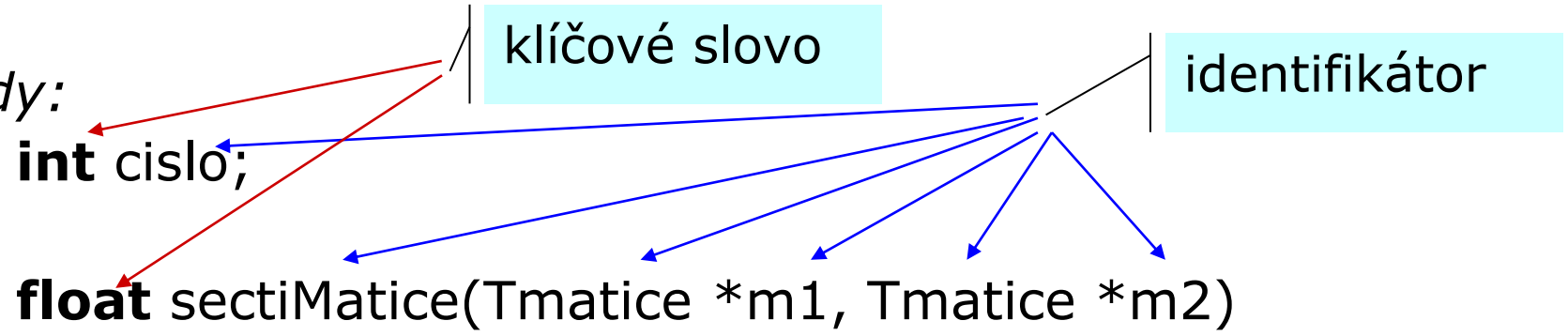
identifier-nondigit:
nondigit

nondigit: one of
_ a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

Identifikátory

- Jazyk C je tzv. *case sensitive* jazyk, to znamená, že rozlišuje velká a malá písmena.
  Příklad: `prom Prom PROM` jsou tři různé identifikátory.
- Znak _ (podtržítko) **nelze** používat libovolně:
  `_prom` - nepoužívat, takto bývají tvořeny systémové identifikátory,
  `prom_x` - používá se poměrně často, zpřehledňuje text,
  `prom_` - nepoužívat, protože se na konci snadno přehlédne.

Příklady:



Identifikátory

Zásady:

- Jména objektů (proměnných, funkcí) psát malými písmeny s využitím _ (podtržítek) nebo kombinace malých a velkých písmen.

  Příklad: `velmi_dlouhy_identifikator`

  `velmiDlouhyIdentifikator`

- Není však vhodné oba způsoby kombinovat v jednom programu. Programátor by si měl vybrat jeden styl a ten potom konzistentně používat.
- Nepoužívat podobné identifikátory, např. `systst` a `syssttst`,
- nepoužívat dva stejné identifikátory odlišené jen typem písma, např. `PROM` a `prom`.

Konstanty (literály)

- Konstanty jsou symboly, reprezentující neměnnou číselnou nebo jinou hodnotu,
- překladač jazyka jim přiřadí typ, který této hodnotě odpovídá.

constant:
integer-constant
floating-constant
enumeration-constant
character-constant

Celočíselné konstanty

- Začíná-li celočíselná konstanta znaky 0x nebo 0X, pak se jedná o zápis v šestnáctkové soustavě se znaky a až f (A až F) ve významu hodnot 10 až 15.
- Začíná-li číslicí 0, pak se jedná o zápis v osmičkové soustavě.

Pozor! Člověk snadno přehlédne, že nejde o desítkové číslo.

- Jinak se jedná o zápis v desítkové soustavě.
- Za celočíselnou konstantou může bezprostředně následovat jedno z písmen l nebo L, které označuje konstantu typu **long**. ANSI C dovoluje použití písmena u nebo U pro označení konstanty bez znaménka.

Celočíselné konstanty [C99]

integer-constant:
decimal-constant integer-suffix$_{opt}$
octal-constant integer-suffix$_{opt}$
hexadecimal-constant integer-suffix$_{opt}$

Příklady :
123 255 45
0173 0377 055
128U 256u
128L 256LL

decimal-constant:
nonzero-digit
decimal-constant digit

nonzero-digit: one of
1 2 3 4 5 6 7 8 9

octal-constant:
0
octal-constant octal-digit

octal-digit: one of
0 1 2 3 4 5 6 7

integer-suffix:
unsigned-suffix long-suffix$_{opt}$
unsigned-suffix long-long-suffix
long-suffix unsigned-suffix$_{opt}$
long-long-suffix unsigned-suffix$_{opt}$

unsigned-suffix: one of
u U
long-suffix: one of
l L
long-long-suffix: one of
ll LL

Celočíselné konstanty [C99]

hexadecimal-constant:
hexadecimal-prefix hexadecimal-digit
hexadecimal-constant hexadecimal-digit

hexadecimal-prefix: one of
0x 0X

hexadecimal-digit: one of
0 1 2 3 4 5 6 7 8 9
a b c d e f
A B C D E F

Příklad : 0x7b 0XFF 0x2D

Zásady:

- Nepoužívat přípony malé l a ll – jsou snadno zaměnitelné s číslicí 1 nebo 11. Použití L nebo LL je přehlednější.

Racionální konstanty (float, double, ...)

- umožňují zapsat číselné konstanty ve formě desetinných čísel.
- vnitřně reprezentovány pomocí mantisy a exponentu, obojí s případným znaménkem.
- zapisují se s desetinnou tečkou, s exponentem nebo oběma způsoby současně.
- interpretují se v desítkové soustavě pokud není uvedeno jinak.
- ANSI C dovoluje koncové písmeno pro rozlišení konstanty typu `float` a `long double`.
- Neuvede-li se přípona, typ konstanty je `double`.

Racionální konstanty [C99]

floating-constant:
decimal-floating-constant
hexadecimal-floating-constant

decimal-floating-constant:
fractional-constant exponent-part$_{opt}$ *floating-suffix*$_{opt}$
digit-sequence exponent-part floating-suffix$_{opt}$

fractional-constant:
digit-sequence$_{opt}$ **.** *digit-sequence*
digit-sequence **.**

exponent-part:
e *sign*$_{opt}$ *digit-sequence*
E *sign*$_{opt}$ *digit-sequence*

sign: one of
+ -

digit-sequence:
digit
digit-sequence digit

floating-suffix: one of
f l F L

Příklad:
.0
1.0
0.
1.0f
3e1
1.0E-3
.00034
1.0e67L

Racionální konstanty

Zásada:

- při porovnávání proměnné typu **float** s konstantou je lepší používat desetinné konstanty.

  if (determinant < 0.0) {…}

Znakové konstanty

- umožňují zapsat jeden znak z používané znakové sady
- zobrazitelné znaky mimo znaků apostrof (') a zpětné (obrácené) lomítko (\) .
- znaky apostrof, zpětné lomítko a tzv. „neviditelné znaky" zapisujeme pomocí jejich znakových kódů (nejčastěji kódů v tabulce ASCII)
- některé často používané znaky (zpětné lomítko, nový řádek, …) mají předdefinovaný zápis pomocí zpětného lomítka ('\\', '\'', '\n', ...)
- Konstanty uvozené znakem \ se také nazývají **escape sekvence** (\ - escape character).

Znakové konstanty

[C99]

character-constant:
' *c-char-sequence* '

c-char-sequence:
c-char
c-char-sequence c-char

c-char:
any member of the source character set except the single-quote **'**, backslash ****, or new-line character
escape-sequence

escape-sequence:
simple-escape-sequence
octal-escape-sequence
hexadecimal-escape-sequence
universal-character-name

simple-escape-sequence: one of
**\' \" \? **
\a \b \f \n \r \t \v

Znakové konstanty [C99]

octal-escape-sequence:
**** *octal-digit*
**** *octal-digit octal-digit*
**** *octal-digit octal-digit octal-digit*

hexadecimal-escape-sequence:
\x *hexadecimal-digit*
hexadecimal-escape-sequence
hexadecimal-digit

Příklad :
'J'
'\112' '\12' '\15'
'\x4A' '\x4a'
'\X4A' - nelze, správně: '\x4A'

Pozor: Neplést s řetězcem – 'A' není totéž co "A"

Řetězcové literály

- Pod pojmem řetězec se v programovacích jazycích myslí posloupnost znaků.
- V jazyce C se řetězce uzavírají do dvojice uvozovek,

  ("Toto je ukázka řetězcové konstanty v C").
- V řetězcových konstantách je možné používat češtinu a znaky s diakritickými znaménky (pozor ale na nastavení jazykového prostředí v operačním systému, např. Windows používají pro češtinu zároveň dvě znakové sady Win1250 a cp852).

Řetězcové literály

[C99]

string-literal:
" *s-char-sequence*$_{opt}$ "

s-char-sequence:
s-char
s-char-sequence s-char

s-char:
any member of the source character set except
the double-quote **"**, backslash ****, or new-line character
escape-sequence

Řetězcové literály

- Velmi často se řetězcová konstanta nevejde na jednu řádku zdrojového kódu. V takovém případě ji můžeme rozdělit na více řádků:

Příklad:

```
"Některé řetězce se těžko vejdou na jednu řádku"
"Některé řetězce se těžko\
 vejdou na jednu řádku"
"Některé řetězce se těžko"
 "vejdou na jednu řádku"
"Některé řetězce se těžko"        "vejdou na"
 "jednu řádku"
"Řetězec může obsahovat i více řádků, které jsou \n
    odděleny znakem nového řádku.\n"
"Je přehlednější to psát takto \n"
```

Řetězcové literály

- Jazyk C nemá žádný standardní datový typ pro přímé uložení řetězce. Je nutné to řešit pomocí pole.
- Řetězec uložený v paměti zabírá vždy o jeden byte více než je počet jeho znaků, protože na konec řetězce je (automaticky) připojen nulový znak ('\0').

Pojmenované konstanty

- Konstanty definované pomocí #define
- Konstanty definované výčtem
- Konstantní proměnné

Konstanty definované pomocí #define

- Konstanty můžeme pojmenovat.
- Konstantám s vhodně zvolenými identifikátory dáváme přednost například před přímým uvedením konstantní hodnoty jako **meze cyklu** nebo **dimenze pole.**
- Modifikace programu pak probíhá velmi snadno změnou hodnoty konstanty.
- Odpadá obtížné uvažování, zdali ta, či jiná hodnota má být modifikována, či nikoliv.
- Pro tyto účely slouží konstanty, které se definují pomocí příkazu preprocesoru #define.

Konstanty definované pomocí #define

Příklady:

```
#define RETEZEC "Konstantni retezec." // zde se nepíše = ani ;
#define PI 3.141592653589793
```

- Tyto konstanty nemají specifikován datový typ.
- Jde pouze o textové zkratky, kdy preprocesor před překladem programu nahradí v textu všechna jména těchto konstant jejich skutečnou hodnotou.

Konstanty definované výčtem

- Definice konstant pomocí datového typu `enum` se používá v případech, kdy spolu mají konstanty nějakým způsobem souviset.
- Používá se například pro specifikaci chybových kódů:

Příklady:

```
enum ErrFileCodes {E_OK, E_BADFILE, E_READ, E_WRITE};
enum ErrValues {EV_NEGATIVE = -1, EV_TOO_BIG = 1024};
```

Konstanty definované výčtem

- Všechny takto definované konstanty jsou typu `int`.
- Při definici jim lze přidělit hodnotu.
- Pokud žádnou hodnotu přidělenu nemají, mají hodnotu od nuly a podle pořadí definic každá další o jedničku větší.

  (E_OK == 0, E_BADFILE == 1, ...)

Konstantní proměnné (od ISO C90)

- Definujeme je pomocí klíčového slova **const** následovaného specifikací datového typu, jménem a inicializací.
- Nejde ovšem o konstanty v pravém slova smyslu, překladač například **nedovolí** její použití při definici pole na **globální úrovni**.
- Jde o proměnné označené modifikátorem **const**, který říká překladači, že má hlídat všechny pokusy o modifikaci.

Příklady:

```
const int KONSTANTA = 123;
const float PLANCK = 6.6256e-34;
const char MALE_A = 'a';
const char *RETEZEC = "Konstantni retezec."
const char RETEZEC[] = "Konstantni retezec." // lepší
```

Konstantní proměnné (od ISO C90)

- Takto můžeme definovat konstantní proměnné všech datových typů.
- Výhodou je, že takto vytvořené konstanty **mají** specifikován datový typ, proto překladač může při jejich použití uplatnit typovou kontrolu.
- Nevýhodou je jejich omezená použitelnost.

Konstantní proměnné (od ISO C90)

Zásady:

- Nepoužívat tzv. „magické konstanty” – nepojmenované konstanty uvnitř kódu.
  - Přispívají ke vzniku chyb a špatné udržovatelnosti kódu.
  - Místo nich je lepší používat pojmenované konstanty (#define nebo konstantní proměnné).
- Názvy konstant se píší vždy velkými písmeny, aby je šlo v kódu odlišit od běžných identifikátorů.
- Kde to jde, používat konstantní proměnné, pro významově spřízněné celočíselné konstanty používat výčtový typ, v ostatních případech #define konstanty.

Interpunkce

[C99]

punctuator: one of

```
[ ] ( ) { } . ->
++ -- & * + - ~ !
/ % << >> < > <= >= == != ^ | && ||
? : ; ...
= *= /= %= += -= <<= >>= &= ^= |=
, # ##
<: :> <% %> %: %:%:
```

- Norma myslí i na uživatele, kteří mají speciální klávesnice, kde chybí znaky jako [ ],{ },# ##
- Pro tyto účely je možné využívat kombinace <: :> , <% %> , %: %:%:
- Obecně se ale nedoporučuje je používat, protože to je pro většinu programátorů velmi nezvyklé a takový kód je potom prakticky nečitelný.

Komentáře

- Komentář je text, který můžeme napsat kamkoliv do zdrojového kódu a nemá žádný vliv na překlad.
- Nejčastěji se používá kvůli vysvětlení určité části kódu.
- Vhodné používání je silně doporučeno a patří k dobré programátorské kultuře.
- Komentář se může vyskytnout všude tam, kde je povolen bílý znak.

Komentáře

Příklad :

```
/* Ukázka komentáře */
// komentář do konce řádku (podle nové normy)
// komentář do konce řádku ať je v něm cokoli /* */ // ...

Není možné vnořovat komentáře /* */:
/* Ukázka komentáře, v němž je vnořen /*další komentář*/ takhle
nelze*/

/*  Ale tohle lze: // stále uvnitř komentáře */

// /* v komentáři */  stále v komentáři  -- lze
```

Komentáře

Doporučení:

- pište komentáře průběžně při psaní zdrojového kódu a ne až nakonec až zbude čas,
- pište komentář hned na začátek každého zdrojového souboru a uveďte sem název programu, jméno autora, datum a verzi.
- Vytvářejte funkce od úvodního komentáře – ujasníte si, co od vytvářené funkce očekáváte.

Komentáře

Příklad :

```
/*****************************************************************/
/* * *                Práce s maticemi v jazyce C                * * */
/* * *                                                           * * */
/* * *                        Verze:1                            * * */
/* * *                                                           * * */
/* * *                     Jaroslav  Nový                        * * */
/* * *                       říjen 2019                          * * */
/*****************************************************************/
```

Pozn.:

Existují systémy, které umí z komentářů ve zdrojovém souboru vygenerovat hypertextovou programátorskou dokumentaci (např. Doxygen pro C, C++, javadoc pro Javu). Proto se vyplatí psát dostatečně kvalitní komentáře už v průběhu psaní kódu.

Komentáře

Styl vhodný pro zpracování Doxygenem:

```
/**
 * Funkce počítá součet dvou matic stejných
 * rozměrů. Výsledek ukládá do …
 */
```

Proměnné a deklarace

Datový objekt – obecné označení jakéhokoliv údaje uloženého v operační paměti.

!!! Nezaměňovat s pojmem *objekt* v objektově orientovaném programování !!!.

Při programování potřebujeme uchovávat data.

Mohou to být:

- údaje zadané uživatelem (jeho jméno, věk, adresa, posloupnost čísel, kterou chce uživatel seřadit, …),

Proměnné

- údaje načtené z diskového souboru (textové, číselné, …),
- různé dočasné a pomocné hodnoty (počet opakování),
- operandy, výsledky operací.

- Paměťová místa ve kterých uchováváme data označujeme jako **proměnné**.
- Proměnná je datový objekt určitého typu, hodnota tohoto objektu se může za běhu programu měnit.

Proměnné a deklarace

Deklarace proměnné je konstrukce, která přidělí proměnné jméno a typ, ale nevytvoří ji. Najdeme je někdy v hlavičkových souborech, i když jejich použití tímto způsobem je neobvyklé.
Definice proměnné je v jazyce C současně deklarací a kromě jména a datového typu přidělí proměnné i paměťový prostor.

Příklady:
```
int i, j;
char znak;
float podil,odmocnina;
```

Inicializace proměnné
Pokud již při definování proměnné víme, jakou bude mít počáteční hodnotu, můžeme ji inicializovat.

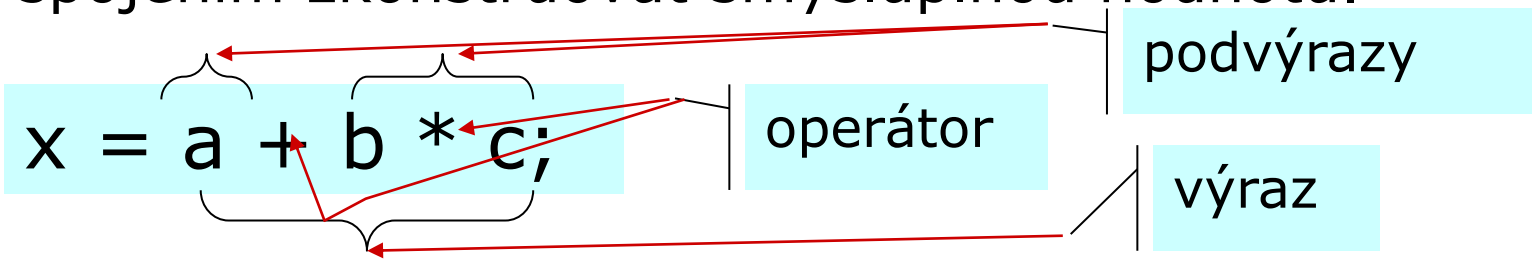
Příklady:
```
char znak = 'a';
int rozmer = 100;
```

Operátory a výrazy

- **Výraz** – konstrukce jazyka, která má hodnotu (nějakého datového typu).
- **Operand** – je částí výrazu na kterou je aplikován jeden z legálních operátorů.
  - Má také hodnotu konkrétního datového typu.
  - Někdy se operandům říká podvýrazy, protože operandem může být i složitější výraz, např. (a+b)*(c+d).

Operátory a výrazy

- **Operátor** – určuje, jakým způsobem se z operandů získá hodnota výrazu.
  - Pořadí vyhodnocování podvýrazů ve výrazu je dáno prioritami jednotlivých operátorů nebo závorkami, pokud byly použity.
  - Datové typy operandů určují, které operátory je možné v daném okamžiku použít.
  - Na konkrétním typovém systému jazyka záleží, zda se aplikace operátoru na různé datové typy vyhodnotí jako chybná konstrukce nebo zda je možno tímto spojením zkonstruovat smysluplnou hodnotu.



Operátory a výrazy

- **Výraz x příkaz**
  - Rozdíl
    - výraz nabývá dále použitelné hodnoty,
    - primárním úkolem příkazu je vykonat nějaký kód.
  - Pokud je výsledkem kódu v příkazu nějaká hodnota a nejde o výraz s přiřazením, hodnota se zahodí (nepoužije, bude se ignorovat).
  - V jazyce C se z výrazu stane příkaz tím, že jej ukončíme středníkem (samozřejmě v místě, kde to dává smysl - uprostřed podmínky cyklu to samozřejmě nejde).
  - Samotný středník (;) představuje prázdný příkaz (null statement), který se někdy používá například v cyklech.

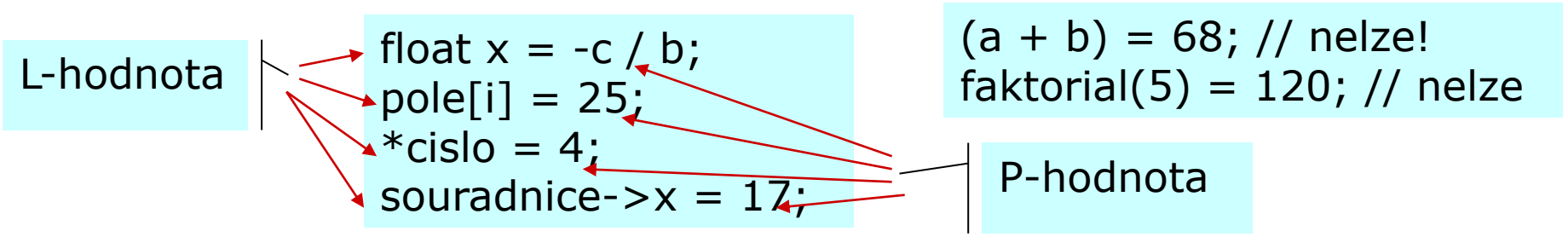
```
x = 64 // výraz s přiřazením
x = 64; // příkaz
printf("Hello world!"); // příkaz
a + b; // příkaz – výsledek výrazu se "zahodí"/nepoužije
```

"mrtvý kód"

Operátory a výrazy

Operátor přiřazení, L-hodnota a P-hodnota

- Operátor přiřazení (=) kopíruje hodnotu výrazu na své pravé straně do proměnné na levé straně.
- V této souvislosti se mluví o L-hodnotě a P-hodnotě (anglicky L-value a R-value).
- **L-hodnota** – je objekt v paměti, kterému lze přiřadit hodnotu.
  - proměnná, prvek pole či struktury nebo paměť odkazovaná přes ukazatel.
- **P-hodnota** – výraz, který má vždy hodnotu a který vystupuje na pravé straně přiřazovacího operátoru.



Operátory a výrazy

- **Přiřazení**
  - V jazyce C zavedeno jako operátor (=).
  - V praxi to znamená, že operace přiřazení má vlastní hodnotu, kterou lze použít dále ve složitějším výrazu.
  - Může se tedy vyskytovat i v P-výrazech.

používat opatrně

Příklad:

```
x1 = x2 = -c/b;
```

P-hodnota

Operátory a výrazy

- Jazyk C ještě poskytuje celou kolekci rozšířených přiřazovacích operátorů, které zjednodušují zápis často používaných výrazů typu:

  L-hodnota = L-hodnota *operátor* výraz;

- V jazyce C jsou definovány tyto rozšířené přiřazovací operátory:

  +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=

Příklad:

x *= a + b; je totéž jako x = x * (a + b);

x * = c + b;

// syntaktická chyba – mezi operátorem a = nesmí být mezera

Aritmetické operátory

- V jazyce C definovány nad číselnými datovými typy.
- Tři skupiny:
  - unární,
  - binární aritmetické operátory,
  - speciální unární operátory.

Aritmetické operátory

[HePa13]

- unární,

Unární plus	Unární mínus
+	-

  - Ke změně znaménka výrazu.
  - Lze použít jak s celočíselnými tak i s racionálními typy.

Příklad:

```
i = -5;
```

Aritmetické operátory

[HePa13]

- binární aritmetické operátory,

sčítání	odčítání	násobení	dělení	modulo
+	-	*	/	%

  - Operátor modulo (%)
    - zbytek po celočíselném dělení
    - jen s celočíselnými typy.

Pozor! V jazyce C může operace modulo produkovat i záporné hodnoty (-2%4 == -2).

Příklad:

```
x = a * b;
```

Aritmetické operátory

- Operátory (+ - * /) fungují kontextově podle datového typu operandů.

`int` op `int` … celočíselné dělení – výsledkem je `int`
`int` op `float` … dělení v pohyblivé řádové čárce – výsledkem je `float`
`float` op `int` … dělení v pohyblivé řádové čárce – výslekem je `float`
`float` op `float` … dělení v pohyblivé řádové čárce – výsledkem je `float`

- Multiplikativní operátory (*, /, %) mají vyšší prioritu než aditivní operátory (+, -), takže matematické výrazy můžeme zapisovat přirozeným způsobem:

```
x = 10 + 2*3; // výsledkem je 16
```

Aritmetické operátory

[HePa13]

- speciální unární operátory,

Unární přičtení jedničky	Unární odečtení jedničky
++	--

  - Oba existují jako předpony (prefix) a jako přípony (suffix). Obě verze se liší svým významem

L-hodnota++

  - inkrementace po použití hodnoty
  - nejprve vrátí původní hodnotu (pokud je použit ve výrazu) a poté přičte k L-hodnotě jedničku

```
i++; // inkrementace proměnné i
```

Aritmetické operátory

[HePa13]

++ L-hodnota

- inkrementace před použitím hodnoty.
- nejprve přičte jedničku a teprve potom vrátí hodnotu výsledku (pokud je použit ve výrazu),
- tyto operátory lze použít výhradně na L-hodnotu, protože do ní ukládají výsledek.

28++ … nelze
--(a + b) … nelze

i = i++; // POZOR! Nedefinovaná hodnota – není jasné, v jakém pořadí se bude modifikovat hodnota proměnné i

Logické a relační operátory

- Pracují s logickými hodnotami.
- V jazyce C produkují výsledky typu **int**
  - C interpretuje hodnotu 0 jako nepravdu,
  - nenulovou hodnotu jako pravdu.
- ISO C99 – datový typ **bool** (**false, true)**
  - kompatibilní s typem **int**
  - nutno dovézt rozhraní <stdbool.h>
  - identifikátor **true** je definován jako konstanta s hodnotou 1
    - Nikdy nedělejte toto: if (vyraz == true)

Logické a relační operátory [HePa13]

Logický součin	Logický součet	Logická negace
&&	||	!

Rovnost	Nerovnost	Menší než	Menší nebo rovno	Větší než	Větší nebo rovno
==	!=	<	<=	>	>=

Zkrácené vyhodnocování výrazů

- Jazyk C používá zkrácené vyhodnocování (short circuit) logických výrazů.

Logické a relační operátory

- Logické výrazy se vyhodnocují zleva doprava a jakmile je jistý výsledek, vyhodnocování se ukončí.

```
if (x != 0 && y / x < z)
```

- Zde nedojde k dělení nulou, protože pokud x je rovno nule, po vyhodnocení podvýrazu x != 0, je jasný výsledek celého výrazu a druhá část už se nebude vyhodnocovat.
- V jazyce C je priorita aritmetických a relačních operátorů vyšší než priorita logických operátorů => logické výrazy není třeba přehnaně závorkovat.

Bitové operátory

[HePa13]

- Pro práci s čísly na úrovni jednotlivých bitů.
- Jazyk C poskytuje šest bitových operátorů.
- Tyto operátory mají odlišnou funkci od logických operátorů!

Bitový součin (AND)	Bitový součet (OR)	Bitová negace (NOT)	Bitový exkluzivní součet (XOR)	Bitový posuv vlevo	Bitový posuv vpravo
&	|	~	^	<<	>>

Bitové operátory

- Výsledkem je číselná hodnota!
- Výsledkem není logická hodnota!

Příklady:

```
0xF0 && 0x88  // logický součin – výsledkem je true (1)

0xF0 & 0x88 // bitový součin – výsledkem je 0x80
```

Adresové operátory

[HePa13]

- Ve spojení s ukazateli.

Referenční operátor	Dereferenční operátor
&jmeno	**ukazatel*

- Referenční operátor - vrací adresu proměnné.
- Dereferenční operátor - vrací hodnotu paměťového místa, odkazovaného ukazatelem.

Příklady:

```
int *uCislo = &cislo;
// (int *) typ ukazatel na int, ne dereferenční operátor
int hodnota = *uCislo; // nyní jde o dereferenční operátor
```

Operátor přetypování [HePa13]

(typ)výraz

- Pro explicitní změnu datového typu výrazu.
- Programátor přebírá zodpovědnost za chování typového systému jazyka.
- Nevhodné použití explicitních konverzí způsobuje problémy!
- Programátor by si měl být vědom, jaké dopady bude mít tato násilná změna datového typu.

Operátor přetypování

- Často používán s ukazateli, zejména s obecným ukazatelem (void *).

Příklady:
(char)ordinální_hodnota … získání znaku z ordinální hodnoty

(int)výraz_float … oříznutí desetinné části (Pozor! Problémy s velkými čísly. Raději trunc(), round(), …)

(float)výraz_int … převod na racionální typ

Operátor sizeof

[HePa13]

sizeof(typ)

sizeof(výraz)

- Vrací počet bajtů, které zabírá konkrétní datový typ nebo výraz.
- Norma jazyka nezaručuje přesnou velikost číselných datových typů!

Operátor sizeof

- Operátor vrací velikost v bajtech.

Příklady:
sizeof(int) // 2, 4 nebo 8 – záleží na procesoru
sizeof(promenna) // rozměr datového typu proměnné

- Zvláštní význam má u polí – vrací součet velikostí jeho jednotlivých položek.
  - Ale ne u polí tvořených přes ukazatel!

Příklad:
```
int pole[10];
sizeof(pole) // == 10*sizeof(int)
             // == 40 – pokud sizeof(int) == 4
```

Podmíněný operátor (ternární)

- Pro vytváření výrazů, jejichž výsledek závisí na vstupní podmínce.
- Syntaxe:

  podmínka **?** *varianta_true* **:** *varianta_false*
- Pokud je podmínka pravdivá, má výraz hodnotu *varianta_true*, jinak má hodnotu *varianta_false*.

Příklad:

```
int max = (a > b)? a : b; // maximum z a, b
```

!! Co když se a rovná b?

Podmíněný operátor (ternární)

Doporučení:

- Používat pouze ve výrazech.
- Nezneužívat jako náhradu příkazu if.
- Nevnořovat ternární operátory do sebe – vede k nesrozumitelnému kódu.

Operátor volání funkce

jméno_funkce()

jméno_funkce(parametry)

- V jazyce C je pro zavolání funkce nutné použít operátor volání funkce, a to i pro funkce, které nemají žádné parametry.
- Identifikátor funkce bez závorek má význam ukazatele na tuto funkci.

Příklad:

```
getchar();
printf("HELLO.");
```

Přístupové operátory

[HePa13]

Indexování	Přístup k prvku struktury	Přístup k prvku struktury přes ukazatel na strukturu
[]	**.**	**->**

Příklad:

```
pole[i+2] = pole[i+1]+pole[i];
souradnice.x = 10;
souradnice.y = 20;
```

(*ukazatel).prvek ekvivalentní s ukazatel->prvek

Operátor čárka

Výraz, výraz

- Vyhodnocuje se zleva doprava a celkový výraz nabývá hodnoty nejpravějšího podvýrazu.
- Používá se například v příkazu `for`.

!! zde to není operátor přiřazení

Příklad:

```
for (int i = 0, j = 10; i != j; i++, j--) // i++, j-- - použití operátoru čárka

int vysledek = (a+b, c+d); // výsledkem je c + d, a+b se zahodí
vysledek = a+b , c+d;   // výsledkem je a+b, c+d je mrtvý kód
```

priorita "=" > priorita ","

Zásada:

raději nepoužívat, smysluplně se dá použít jen v minimálním počtu případů.

Priorita operátorů

- Udává pořadí vyhodnocení podvýrazů.
- Vhodně zavedená hierarchie priorit operátorů dovoluje značně redukovat počet závorek ve výrazech.
- Pomocí závorek lze měnit pořadí vyhodnocení výrazu podle potřeby.
- Ve všech jazycích je přesně specifikována priorita všech operátorů, (v některých jsou však priority navrženy neprakticky – viz Pascal).
- Platí pravidlo:

"Nejsi-li si jistý prioritou operátorů, závorkuj."

Příklad:

```
X = 3+2*10; // výsledek 23
x = (3+2) * 10; // výsledek 50
```

Priorita operátorů

[HePa13]

Priorita	**Operátory**	**Asociativita**	**skupina op.**
1.	() [] -> .	zleva doprava	primární
2.	! ~ ++ -- + - (typ) * & **sizeof**	zprava doleva	unární
3.	* / %	zleva doprava	multiplikativní
4.	+ -	zleva doprava	aditivní
5.	<< >>	zleva doprava	posuny
6.	< <= > >=	zleva doprava	relační
7.	== !=	zleva doprava	relační
8.	&	zleva doprava	bitový součin
9.	^	zleva doprava	exkl. bit. souči.
10.	|	zleva doprava	bitový součet
11.	&&	zleva doprava	logický součin
12.	||	zleva doprava	logický součet
13.	?:	zprava doleva	ternární podm.
14.	= += -= *= /= %= >>= <<= &= |= ^=	zprava doleva	přiřazení
15.	,	zleva doprava	op. čárka

Priorita operátorů

[HePa13]

Priorita	**Operátory**	**Asociativita**	**skupina op.**
1.	() [] -> .	zleva doprava	primární
2.	! ~ ++ -- + - (typ) * & **sizeof**	**zprava doleva**	unární
3.	* / %	zleva doprava	multiplikativní
4.	+ -	zleva doprava	aditivní
5.	<< >>	zleva doprava	posuny
6.	< <= > >=	zleva doprava	relační
7.	== !=	zleva doprava	relační
8.	&	zleva doprava	bitový součin
9.	^	zleva doprava	exkl. bit. souči.
10.	|	zleva doprava	bitový součet
11.	&&	zleva doprava	logický součin
12.	||	zleva doprava	logický součet
13.	?:	**zprava doleva**	ternární podm.
14.	= += -= *= /= %= >>= <<= &= |= ^=	**zprava doleva**	přiřazení
15.	,	zleva doprava	op. čárka

Priorita operátorů

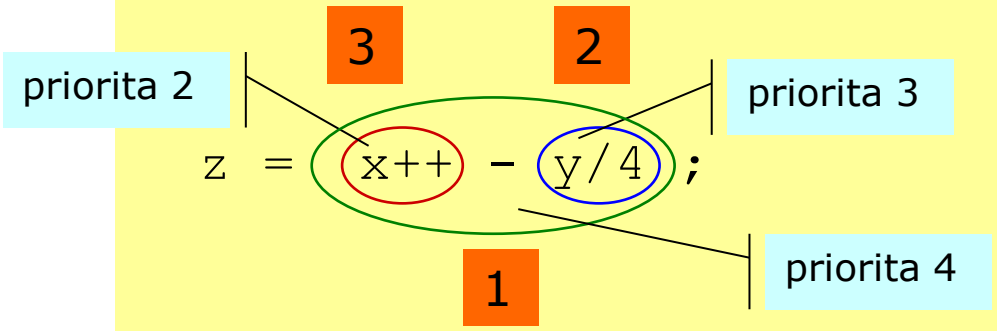
- Čím nižší číslo skupiny, tím vyšší priorita

Příklad:

```
#include <stdio.h>
int main(void)
{
  signed int z, x=3, y=9;

  z = x++ - y/4;

  printf("%d \n",z);
  return 0;
}
```



Principy vyšších programovacích jazyků

Kontrolní otázky

1. Čím je určen datový typ?
2. Uveďte rozdělení datových typů.
3. Vysvětlete použití specifikátoru typu *signed* a *unsigned*.
4. Jak překladač rozliší typ konstanty (literálu)?
5. Uveďte druhy operátorů.
6. K čemu slouží operátor *sizeof*?
7. Jak probíhá vyhodnocení výrazů?

Úkoly k procvičení

Zadání úkolů:

- Zapište v jazyce C:
  - podmínky: $x \in <0,1)$, $x \notin (5,10>$, $x \in \{2,5,8\}$, $x \notin \{1,5\}$, $x \neq \pm 1$, c je dělitelné 5, b je sudé, $x \geq 1 \wedge x$ je liché, c je dělitelné 2 nebo 3.
  - negaci výrazů (bez použití operátoru (!)): $A>0$, $A+B \geq C$, $X*Z=Z*A$, $(A>0) \wedge (B>0)$, $(A=0) \vee (B=0)$, $\neg(A>B)$,