

Řídicí struktury, výrazy a funkce

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 03

BAB36PRGA – Programování v C

Přehled témat

■ Část 1 – Řídicí struktury

Příkaz a složený příkaz

Příkazy řízení běhu programu

Konečnost cyklu

S. G. Kochan: kapitoly 5 a 6

■ Část 2 – Výrazy

Výrazy a operátory

Přīřazení

S. G. Kochan: kapitola 4, 12

■ Část 3 – Zadání 2. domácího úkolu (HW2)

Část I

Řídicí struktury

Příkaz a složený příkaz (blok)

- Příkaz je výraz zakončený středníkem.

Příkaz tvořený pouze středníkem je prázdný příkaz.

- Blok je tvořen seznamem definic proměnných a příkazů.

- Uvnitř bloku, definice proměnných zpravidla předchází příkazům.

Záleží na standardu jazyka, platí pro ANSI C (C89, C90).

- Začátek a konec bloku je vymezen složenými závorkami { a }.

- Bloky mohou být vnoreny do jiného bloku.

```
void function(void)
{ /* function block start */
  /* inner block */
  for (i = 0; i < 10; ++i)
  {
    //inner for-loop block
  }
}

void function(void) { /* function block start */
  { /* inner block */
    for (int i = 0; i < 10; ++i) {
      //inner for-loop block
    }
  }
}
```

Různé kódovací konvence.

Srozumitelnost, čitelnost kódu - kódovací konvence a styl (čistota kódu)

- Konvence a styl je důležitý, protože podporuje přehlednost a čitelnost.

https://www.gnu.org/prep/standards/html_node/Writing-C.html

- Formátování patří k úplným základem. *Nastavte si automatické formátování v textovém editoru.*

- Volba výstředního jména identifikátorů podporuje čitelnost.

Co může být jasné nyní, za pár dní či měsíců může být jinak.

- **Cvičte se v kódovací konvenci a zvoleném stylu i za cenu zdánlivě pomalejšího zápisu kódu. Přehlednost je důležitá, zvláště pokud hledáte chybu.**

Nezřídka je užitečné nebýt se začít úplně znovu a lépe.

- Doporučená konvence v rámci PRGA

```
1 void function(void)
2 { /* function block start */
3   for (int i = 0; i < 10; ++i) {
4     //inner for-loop block
5     if (i == 5) {
6       break;
7     }
8   }
9 }
```

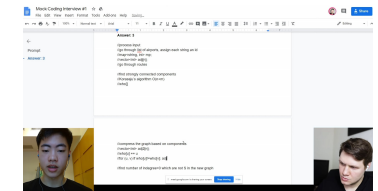
Osobní preference přednášejících: odsazení 3 znaky, mezery místo tabulátorů.

Srozumitelnost a čitelnost kódu - kódovací konvence

- Existují různé kódovací konvence; inspiřujte se existujícími doporučeními a čtením reprezentativních kódů.



Clean Code - Uncle Bob / Lesson 1
<https://youtu.be/7EmboKQH81M>



Google Coding Interview with a High School Student
<https://youtu.be/qz9tKlF431k>

<http://users.ece.cmu.edu/~eno/coding/CodingStandard.html>;
<https://www.doc.ic.ac.uk/lab/cplusc/cstyle.html>;
http://en.wikimedia.org/wiki/Indent_style;
<https://google.gisthub.com/styleguide/cppguide.html>;
<https://www.kernel.org/doc/Documentation/process/coding-style.rst>

Složený příkaz a zanořování 1/2

Čtyři úrovně zanořování.

```
1 int get_sum_of_even_numbers(int from, int to)
2 {
3   if (from < to) {
4     int sum = 0;
5     for (int number = from; number <= to; ++number) {
6       if (number % 2 == 0) {
7         sum += number;
8       } // end for loop
9     }
10    return sum;
11  } else {
12    return 0;
13  }
14 }
```

Vyjmutí (definice nové funkce).

```
1 int filter_odd(int number);
2
3 int get_sum_of_even_numbers(int from, int to)
4 {
5   if (from < to) {
6     int sum = 0;
7     for (int number = from; number <= to; ++number) {
8       sum += filter_odd(number);
9     } // end for loop
10    return sum;
11  } else {
12    return 0;
13  }
14 }
```

Mířme na čitelnější podobu.

```
1 int get_sum_of_even_numbers(int from, int to)
2 {
3   if (from > to) return 0;
4   int sum = 0;
5   for (int number = from; number <= to; ++number) {
6     sum += filter_odd(number);
7   } // end for loop
8   return sum;
9 }
```

- Použitím technik **vyjmutí** a inverze redukuje počet zanořování. <https://youtu.be/CFR8duuJG-4>

Složený příkaz a zanořování 2/2

Inverze (záměna podmínky hodnoty vstupu).

```
1 int filter_odd(int number);
2
3 int get_sum_of_even_numbers(int from, int to)
4 {
5   if (from > to) {
6     return 0;
7   }
8   int sum = 0;
9   for (int number = from; number <= to; ++number) {
10    sum += filter_odd(number);
11  } // end for loop
12  return sum;
13 }
14
15 int filter_odd(int number)
16 {
17   if (number % 2 == 0) {
18     return number;
19   }
20   return 0;
21 }
```

Finální „zkompaktnění“.

```
1 int filter_odd(int number);
2
3 int get_sum_of_even_numbers(int from, int to)
4 {
5   if (from > to) return 0;
6
7   int sum = 0;
8   for (int number = from; number <= to; ++number) {
9     sum += filter_odd(number);
10  } // end for loop
11  return sum;
12 }
13
14 int filter_odd(int number)
15 {
16   return (number % 2 == 0) ? number : 0;
17 }
```

- Použitím technik **vyjmutí** a **inverze** redukuje počet zanořování. <https://youtu.be/CFR8duuJG-4>

Příkazy řízení běhu programu

- Podmíněné řízení běhu programu
 - Podmíněný příkaz: `if ()` nebo `if () ... else`
 - Programový přepínač: `switch () case ...`

- Cykly

- `for ()`
- `while ()`
- `do ... while ()`

- Nepodmíněné větvení programu

- `continue`
- `break`
- `return`
- `goto`

Podmíněné větvení – if

- if (výraz) příkaz1; else příkaz2
- Je-li hodnota výrazu `výraz != 0 (TRUE)`, provede se příkaz `příkaz1` jinak `příkaz2`.
- Část `else` je nepovinná. *Příkaz může být blok příkazů.*
- Podmíněné příkazy mohou být vnořené a můžeme je řetězit.

```

int max;
if (a > b) {
    if (a > c) {
        max = a;
    }
}

```

```

int max;
if (a > b) {
    ...
} else if (a < c) {
    ...
} else if (a == b) {
    ...
} else {
    ...
}

```

- Příklad zápisu

```

1 if (x < y) {
2     int tmp = x;
3     x = y;
4     y = tmp;
5 }

```

```

1 if (x < y) {
2     min = x;
3     max = y;
4 } else {
5     min = y;
6     max = x;
7 }

```

Jaký je smysl těchto programů?

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 12 / 57

Příkaz větvení switch

- Příkaz `switch` (přepínač) umožňují větvení programu do více větví na základě různých hodnot výrazu výtčového (celočíslného) typu, jako jsou např. `int`, `char`, `short`, `enum`.
- Základní tvar příkazu.

```

switch (výraz) {
    case konstanta1: příkazy1; break;
    case konstanta2: příkazy2; break;
    ...
    case konstanta_n: příkazy_n; break;
    default: příkazy_def; break;
}

```

kde *konstanty* jsou téhož typu jako *výraz* a *příkazy*; jsou posloupnosti příkazů.
Sémantika: vypočte se hodnota výrazu a provedou se ty příkazy, které jsou označeny konstantou s identickou hodnotou. Není-li vybrána žádná větev, provedou se příkazy_def (pokud jsou uvedeny).

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 13 / 57

Programový přepínač – switch

- Přepínač `switch(výraz)` větví program do *n* směrů.
- Hodnota `výraz` je porovnávána s *n* konstantními výrazy typu `int` příkazy. `case konstanta1: ...`
- Hodnota `výraz` musí být celočíselná a hodnoty `konstanta_i` musejí být navzájem různé.
- Pokud je nalezena shoda, program pokračuje od tohoto místa dokud nenajde příkaz `break` nebo konec příkazu `switch`.
- Pokud shoda není nalezena, program pokračuje nepovinnou sekcí `default`. *Sekce default se zpravidla uvádí jako poslední.*
- Příkazy `switch` mohou být vnořené.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 14 / 57

Programový přepínač switch – Příklad

```

switch (v) {
    case 'A':
        printf("Upper 'A'\n");
        break;
    case 'a':
        printf("Lower 'a'\n");
        break;
    default:
        printf("It is not 'A' nor 'a'\n");
        break;
}

```

```

if (v == 'A') {
    printf("Upper 'A'\n");
} else if (v == 'a') {
    printf("Lower 'a'\n");
} else {
    printf("It is not 'A' nor 'a'\n");
}

```

lec03/switch.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 15 / 57

Větvení switch – pokračování ve vykonávání dalších větví

- Příkaz `break` dynamicky ukončuje větev, pokud jej neuvedeme, pokračuje se v provádění další větve.

Příklad volání více větví

```

1 int part = ?
2 switch(part) {
3     case 1:
4         printf("Branch 1\n");
5         break;
6     case 2:
7         printf("Branch 2\n");
8     case 3:
9         printf("Branch 3\n");
10        break;
11     case 4:
12        printf("Branch 4\n");
13        break;
14    default:
15        printf("Default branch\n");
16        break;
17 }

```

- part ← 1 Branch 1
- part ← 2 Branch 2
- part ← 3 Branch 3
- part ← 4 Branch 4
- part ← 5 Default branch

lec03/demo-switch_break.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 16 / 57

Příklad větvení switch vs if-then-else

- Napište konverzní program, který podle čísla dnu v týdnu vytiskne na obrazovku jméno dne. Ošetřete případ, kdy bude zadané číslo mimo platný rozsah (1 až 7).

Příklad implementace

```

int day_of_week = 3;
switch (day_of_week) {
    case 1:
        printf("Monday");
        break;
    case 2:
        printf("Tuesday");
        break;
    case 7:
        printf("Sunday");
        break;
    default:
        fprintf(stderr, "Invalid number");
        break;
}

```

```

if (day_of_week == 1) {
    printf("Monday");
} else if (day_of_week == 2) {
    printf("Tuesday");
} else ... {
} else if (day_of_week == 7) {
    printf("Sunday");
} else {
    fprintf(stderr, "Invalid number");
}

```

Oba způsoby jsou sice funkční, nicméně elegantněji lze vyřešit úlohu použitím datové struktury pole nebo ještě lépe asociativním polem / hash mapou.

lec03/demo-switch_day_of_week.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 17 / 57

Cykly

- Cyklus `for` a `while` testuje podmínku opakování před vstupem do těla cyklu.
 - `for` – inicializace, podmínka a změna řídicí proměnné.
- `while` – řídicí proměnná v těle cyklu.

```

for (int i = 0; i < 5; ++i) {
    ...
}

```

```

while (i < 5) {
    ...
    i++;
}

```

```

int i = -1;
do {
    ...
    i++;
} while (i < 5);

```

Ekvivalentní provedení pěti cyklů.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 18 / 57

Cyklus while a do-while

- Základní příkaz cyklu `while` má tvar `while (podmínka) příkaz`.
- Základní příkaz cyklu `do-while` má tvar `do příkaz while (podmínka)`.

Příklad

```

q = x;
while (q >= y) {
    q = q - y;
}

```

```

q = x;
do {
    q = q - y;
} while (q >= y);

```

- Jaká je hodnota proměnné *q* po skončení cyklu pro hodnoty.
 - `x ← 10 a y ← 3`
 - `x ← 2 a y ← 3`

while: 1, do-while: 1

while: 2, do-while: -1

lec03/demo-while.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 19 / 57

Cyklus for

- Základní příkaz cyklu `for` má tvar `for (inicializace; podmínka; změna) příkaz`.
- Odpovídá cyklu `while` ve tvaru: `inicializace; while (podmínka) { příkaz; změna; }`
- Změnu řídicí proměnné lze zkráceně zapsat operátorem inkrementace nebo dekrementace `++` a `--`.
- Alternativně lze též použít zkrácený zápis přiřazení, např. `+=`.

Příklad

```

for (int i = 0; i < 10; ++i) {
    printf("i: %i\n", i);
}

```

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 20 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Cyklus for(; ;)

- Príkaz **for** cyklu má tvar `for ([vyraz1]; [vyraz2]; [vyraz3]) prikaz;`
- Cyklus **for** používá řídicí proměnnou a probíhá následovně:
 - `vyraz1` – Inicializace (zpravidla řídicí proměnné);
 - `vyraz2` – Test řídicího výrazu;
 - Pokud `vyraz2 != 0` provede se `prikaz`, jinak cyklus končí;
 - `vyraz3` – Aktualizace proměnných na konci běhu cyklu;
 - Opakování cyklu testem řídicího výrazu.
- Výrazy `vyraz1` a `vyraz3` mohou být libovolného typu.
- Libovolný z výrazů lze vynechat.
- break** – cyklus lze nuceně opustit příkazem `break`.
- continue** – část těla cyklu lze vynechat příkazem `continue`.

Príkaz přeruší vykonávání těla (blokového příkazu) pokračuje vyhodnocením vyraz₃.

- Při vynechání řídicího výrazu `vyraz2` se cyklus bude provádět nepodmíněně.


```
for (;) {...}
```

Nekonečný cyklus

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 21 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Príkaz continue

- Príkaz návratu na vyhodnocení řídicího výrazu – **continue**.
- Príkaz **continue** lze použít pouze v těle cyklů.
 - `for ()`
 - `while ()`
 - `do...while ()`
- Príkaz **continue** přeruší vykonávání těla cyklu a nově vyhodnocení řídicího výrazu.
- Příklad


```
int i;
for (i = 0; i < 20; ++i) {
    if (i % 2 == 0) {
        continue;
    }
    printf("%d\n", i);
}
```

lec03/continue.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 22 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Předčasné ukončení průchodu cyklu – příkaz continue

- Někdy může být užitečné ukončit cyklus v nějakém místě uvnitř těla cyklu.
 - Například ve vnořených **if** příkazech.
- Príkaz **continue** předepisuje **ukončení průchodu** těla cyklu.

Platnost pouze v těle cyklu!

```
for (int i = 0; i < 10; ++i) {
    printf("i: %i ", i);
    if (i % 3 != 0) {
        continue;
    }
    printf("\n");
}
```

\$ clang demo-continue.c
\$./a.out
i:0
i:1 i:2 i:3
i:4 i:5 i:6
i:7 i:8 i:9
lec03/demo-continue.txt

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 23 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Príkaz break

- Príkaz nuceného ukončení cyklu **break**; lze použít pouze v těle cyklů.
 - `for()`
 - `while()`
 - `do...while()`
- a v těle programového přepínače `switch()`.
- break** způsobí opuštění těla cyklu nebo těla `switch()`.
- Program pokračuje následujícím příkazem, např.


```
int i = 10;
while (i > 0) {
    if (i == 5) {
        printf("i reaches 5, leave the loop\n");
        break;
    }
    i--; // nebo -i; nebo i -= 1; nebo i = i - 1;
    printf("End of the while loop i: %d\n", i);
}
```

lec03/break.c

- Z hlediska přehlednosti a čitelnosti je vhodné změnu řídicí proměnné realizovat na konci cyklu.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 24 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Předčasné ukončení vykonávání cyklu – příkaz break

- Príkaz **break** předepisuje ukončení cyklu.

Program pokračuje následujícím příkazem po cyklu.

```
for (int i = 0; i < 10; ++i) {
    printf("i: %i ", i);
    if (i % 3 != 0) {
        continue;
    }
    printf("\n");
    if (i > 5) {
        break;
    }
}
```

\$ clang demo-break.c
\$./a.out
i:0
i:1 i:2 i:3
i:4 i:5 i:6
lec03/demo-break.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 25 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Príkaz goto

- Príkaz nepodmíněného lokálního skoku **goto** předá řízení na místo určené návěští `navesti` – syntax `goto navesti;`
- Návěští má tvar `navesti prikaz.`

Definice proměnné není příkaz.
- Príkaz **goto** lze použít pouze v těle funkce a skok je možný pouze rámci jediné funkce.


```
int test = 3;
for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 5; ++j) {
        if (j == test) {
            goto loop_out;
        }
        fprintf(stdout, "Loop i: %d j: %d\n", i, j);
    }
}
return 0;
loop_out:
fprintf(stdout, "After loop\n");
return -1;
```

lec03/goto.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 26 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Vnořené cykly

- break** ukončuje vnitřní cyklus.


```
for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
        printf("i-j: %i-%i\n", i, j);
        if (j == 1) {
            break;
        }
    }
}
```
- Vnější cyklus můžeme ukončit příkazem **goto**.


```
for (int i = 0; i < 5; ++i) {
    for (int j = 0; j < 3; ++j) {
        printf("i-j: %i-%i\n", i, j);
        if (j == 2) {
            goto outer;
        }
    }
}
outer:
```

lec03/demo-goto.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 27 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Konečnost cyklů 1/3

- Konečnost algoritmu – pro přípustná data v konečné době skončí.
- Aby byl algoritmus **konečný** musí každý cyklus v něm uvedený skončit po konečném počtu kroků.
- Jedním z důvodů neukončení programu je zacyklení.
 - Program opakovaně vykoná cyklus, jehož podmínka ukončení není nikdy splněna.


```
while (i != 0) {
    j = i - 1;
}
```
 - Cyklus se neprovede ani jednou, nebo neskončí.
 - Záleží na hodnotě `i` před voláním cyklu.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 29 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Konečnost cyklů 2/3

- Základní pravidlo pro konečnost cyklu.
 - Provedením těla cyklu se musí změnit hodnota proměnné použité v podmínce ukončení cyklu.


```
for (int i = 0; i < 5; ++i) {
    ...
}
```
 - Uvedené pravidlo konečnost cyklu nezaručuje.


```
int i = -1;

while ( i < 0 ) {
    i = i - 1;
}
```

Konečnost cyklu závisí na hodnotě proměnné před vstupem do cyklu.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 30 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Konečnost cyklů 3/3

```
while (i != n) {
    ... //příkazy nemenící hodnotu promenne i
    i++;
}
```

lec03/demo-loop_byte.c

- Vstupní podmínka konečnosti uvedeného cyklu.
 - $i \leq n$ pro celá čísla.

*Jak by vypadala podmínka pro proměnné typu double?
Co se stane pokud by proměnná i byla typu unsigned char?*

```
lec03/demo-loop.c
```

- Splnění vstupní podmínky konečnosti cyklu musí zajistit příkazy předcházející příkazu cyklu.
- Zabezpečený program testuje přípustnost vstupních dat.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 31 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Příklad – test, je-li zadané číslo prvočíslem

```
#include <stdbool.h>
#include <math.h>

_Bool isPrimeNumber(int n)
{
    _Bool ret = true;
    for (int i = 2; i <= (int)sqrt((double)n); ++i) {
        if (n % i == 0) {
            ret = false; // leave the loop once if it sure
            break; // n is not a prime number
        }
    }
    return ret;
}
```

lec03/demo-prime.c

- break – po nalezení prvního dělitele nemusíme dále testovat.
- Hodnota výrazu `(int)sqrt((double)n)` se v cyklu nemění.

```
_Bool ret = true; // zbytečně výpočet explicitně opakovat v podmínce
const int maxBound = (int)sqrt((double)n); // opakování těla cyklu
for (int i = 2; i <= maxBound; ++i) {
    ...
}
```

Příklad kompilace spuštění demo-prime.c: `clang demo-prime.c -lm; ./a.out 13`

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 32 / 57

Príkaz a složený příkaz Příkazy řízení běhu programu Konečnost cyklu

Kódovací konvence

- Příkazy **break** a **continue** v podstatě odpovídají příkazům skoku.
- Obecně můžeme říci, že příkazy **break** a **continue** nepřidávají příliš na přehlednosti.
 - Ne nutně break v příkazu switch.*
- Přerušení cyklu **break** nebo **continue** můžeme využít v těle dlouhých funkcí a vnořených cyklech.
 - Ale funkce bychom měli psát krátké a přehledné.*
- Je-li funkce (tělo cyklu) krátké, je význam **break/continue** čitelný.
- Podobně použití na začátku bloku cyklu, např. jako součást testování splnění předpokladů, je zpravidla přehledné.
- Použití uprostřed bloku je však už méně přehledné a může snížit čitelnost a porozumění kódu.

<https://www.scribd.com/doc/38873257/Knuth-1974-Structured-Programming-With-Go-to-Statements>

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 33 / 57

Výrazy a operátory Přířazení

Část II

Výrazy

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 34 / 57

Výrazy a operátory Přířazení

Výrazy

- Výraz předepisuje výpočet hodnoty určitého vstupu.
- Struktura výrazu obsahuje *operandy, operátory a závorky*.
- Výraz může obsahovat:
 - literály;
 - proměnné;
 - konstanty;
 - unární a binární operátory;
 - volání funkcí;
 - závorky.
- Pořadí operací predepsaných výrazem je dáno **prioritou** a **asociativitou** operátorů.

Příklad

```
10 + x * y    poradi vyhodnoceni 10 + (x * y)
10 + x + y    poradi vyhodnoceni (10 + x) + y
```

** má vyšší prioritu než + je asociativní zleva*

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 36 / 57

Výrazy a operátory Přířazení

Výrazy a operátory

- Výraz se skládá z operátorů a operandů.
 - Nejjednodušší výraz tvoří konstanta, proměnná nebo volání funkce.
 - Výraz sám může být operandem.
 - Výraz má **typ** a **hodnotu**. *(Pouze výraz typu void hodnotu nemá.)*
 - Výraz zakončený středníkem ; je příkaz.
- Operátory jsou vyhrazené znaky pro zápis výrazů.
 - Případně posloupnost znaků.*
 - Postup výpočtu výrazu s více operátory je dán prioritou operátorů.
 - Postup výpočtu lze předepsat použitím kulatých závorek (a).*
- Operátory: aritmetické, relační, logické, bitové.
 - Arita operátoru (počet operandů) – unární, binární, ternární.
 - Obecně (mimo konkrétní případy) není pořadí vyhodnocení operandů definováno (nezaměňovat s asociativitou).
 - Např. pro součet f1() + f2() není definováno, který operand se vyhodnotí jako první (jaká funkce se zavolá jako první).*
 - Chování i = ++i + i++; není definováno, závisí na překladači.*
 - Pořadí vyhodnocení je **definováno pro operand v logickém součinu AND a součtu OR**.
 - http://en.cppreference.com/w/c/language/eval_order

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 37 / 57

Výrazy a operátory Přířazení

Základní rozdělení operátorů

- Můžeme rozlišit čtyři základní typy binárních operátorů:
 - Aritmetické operátory – sčítání, odčítání, násobení, dělení;
 - Relační operátory – porovnání hodnot (menší, větší, ...);
 - Logické operátory – logický součet a součin;
 - Operátor přiřazení - na levé straně operátoru = je proměnná (l-hodnota reprezentující místo v paměti).
- Unární operátory:
 - indikující kladnou/zápornou hodnotu: `+` a `-`;
 - operátor – modifikuje znaménko výrazu za něj.*
 - modifikující proměnnou: `++` a `--`;
 - logický operátor doplněk: `!`;
 - bitová negace: `~` (negace bit po bitu).
- Ternární operátor – podmíněný příkaz.
 - Jediný ternární operátor v C je podmíněný příkaz ? :*

http://www.tutorialspoint.com/cprogramming/c_operators.htm

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 38 / 57

Výrazy a operátory Přířazení

Aritmetické operátory

- Operandy aritmetických operátorů mohou být libovolného aritmetického typu.
 - Výjimkou je operátor zbytek po dělení % definovaný pro celá čísla (int).*

*	Násobení	<code>x * y</code>	Součin <code>x</code> a <code>y</code>
/	Dělení	<code>x / y</code>	Podíl <code>x</code> a <code>y</code>
%	Dělení modulo	<code>x % y</code>	Zbytek po dělení <code>x</code> a <code>y</code>
+	Sčítání	<code>x + y</code>	Součet <code>x</code> a <code>y</code>
-	Odčítání	<code>x - y</code>	Rozdíl <code>a</code> a <code>y</code>
+	Kladné znam.	<code>+x</code>	Hodnota <code>x</code>
-	Záporné znam.	<code>-x</code>	Hodnota <code>-x</code>
++	Inkrementace	<code>++x/x++</code>	Inkrementace před/po vyhodnocení výrazu <code>x</code>
--	Dekrementace	<code>--x/x--</code>	Dekrementace před/po vyhodnocení výrazu <code>x</code>

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 39 / 57

Výrazy a operátory Přířazení

Unární aritmetické operátory

- Unární operátory `++` a `--` mění hodnotu svého operandu.
 - Operand musí být l-hodnota, tj. výraz, který má adresu, kde je uložena hodnota výrazu (např. proměnná).*
 - lze zapsat prefixově např. `++x` nebo `--x`;
 - nebo postfixově např. `x++` nebo `x--`;
 - v obou případech se však **liší výsledná hodnota výrazu!**

int i; int a;	hodnota i	hodnota a
<code>i = 1; a = 9;</code>	1	9
<code>a = i++;</code>	2	1
<code>a = ++i;</code>	3	3
<code>a = ++(++i);</code>	nelze, hodnota i++ není l-hodnota	

V případě unárního operátoru i++ je nutné v paměti uchovat původní hodnotu i, a následně inkrementovat hodnotu proměnné i. V případě použití ++i pouze inkrementujeme hodnotu i. Proto může být použití ++i efektivnější.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídicí struktury, výrazy a funkce 40 / 57

Výrazy a operátory Přirazení

Relační operátory

- Operandy relačních operátorů mohou být aritmetického typu, ukazatele shodného typu nebo jeden z nich `NULL` nebo typ `void`.

<code><</code>	Menší než	<code>x < y</code>	1 pro x je menší než y, jinak 0.
<code><=</code>	Menší nebo rovno	<code>x <= y</code>	1 pro x menší nebo rovno y, jinak 0.
<code>></code>	Větší než	<code>x > y</code>	1 pro x je větší než y, jinak 0.
<code>>=</code>	Větší nebo rovno	<code>x >= y</code>	1 pro x větší nebo rovno y, jinak 0.
<code>==</code>	Rovná se	<code>x == y</code>	1 pro x rovno y, jinak 0.
<code>!=</code>	Nerovná se	<code>x != y</code>	1 pro x nerovno y, jinak 0.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 41 / 57

Výrazy a operátory Přirazení

Logické operátory

- Operandy mohou být aritmetické typy nebo ukazatele.
- Výsledek 1 má význam `true`, 0 má význam `false`.
- Ve výrazech `&&` a `||` se vyhodnotí nejdříve levý operand.
- Pokud je výsledek dán levým operandem, pravý se nevyhodnocuje.

Zkrácené vyhodnocování – složité výrazy.

<code>&&</code>	Logické AND	<code>x && y</code>	1 pokud x ani y není rovno 0, jinak 0.
<code> </code>	Logické OR	<code>x y</code>	1 pokud alespoň jeden z x, y není rovno 0, jinak 0.
<code>!</code>	Logické NOT	<code>!x</code>	1 pro x rovno 0, jinak 0.

- Operace `&&` a `||` se vyhodnocují zkráceným způsobem, tj. druhý operand se nevyhodnocuje, pokud lze výsledek určit již z hodnoty prvního operandu.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 42 / 57

Výrazy a operátory Přirazení

Bitové operátory

- Bitové operátory vyhodnocují operandy bit po bitu.

<code>&</code>	Bitové AND	<code>x & y</code>	1 když x i y je rovno 1 (bit po bitu).
<code> </code>	Bitové OR	<code>x y</code>	1 když x nebo y je rovno 1 (bit po bitu).
<code>^</code>	Bitové XOR	<code>x ^ y</code>	1 pokud pouze x nebo pouze y je 1 (exkluzivně právě jedna z variant) (bit po bitu).
<code>~</code>	Bitové NOT	<code>~x</code>	1 pokud x je rovno 0 (bit po bitu).
<code><<</code>	Posun vlevo	<code>x << y</code>	Posun x o y bitů vlevo.
<code>>></code>	Posun vpravo	<code>x >> y</code>	Posun x o y bitů vpravo.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 43 / 57

Výrazy a operátory Přirazení

Příklad – bitových operací

```
uint8_t a = 4;
uint8_t b = 5;

a      dec: 4 bin: 0100
b      dec: 5 bin: 0101
a & b  dec: 4 bin: 0100
a | b  dec: 5 bin: 0101
a ^ b  dec: 1 bin: 0001

a >> 1 dec: 2 bin: 0010
a << 1 dec: 8 bin: 1000
```

[lec03/bits.c](#)
Vizte rekurzivní implementace [lec03/bits-recursive.c](#)

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 44 / 57

Výrazy a operátory Přirazení

Operace bitového posunu

- Operátory bitového posunu posouvají celý bitový obraz o zvolený počet bitů vlevo nebo vpravo.
 - Při posunu vlevo jsou uvolněné bity zleva plněny 0.
 - Při posunu vpravo jsou uvolněné bity zprava:
 - u čísel kladných nebo typu `unsigned` plněny 0;
 - u záporných čísel buď plněny 0 (logický posun) nebo 1 (aritmetický posun vpravo), dle implementace překladače.
- Operátory bitového posunu mají nižší prioritu než aritmetického operátory!
 - `i << 2+1` znamená `i << (2+1)`.

Nebuďte zaskočení nečekanou interpretací – závorkujte!

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 45 / 57

Výrazy a operátory Přirazení

Operátory přístupu do paměti

Zde pro úplnost, více v následujících přednáškách.

- V C lze přímo přistupovat k adrese paměti proměnné, kde je uložena hodnota.
- Přístup do paměti je prostřednictvím ukazatele (`pointeru`).

Dává velké možnosti, ale také vyžaduje zodpovědnost.

Operátor	Význam	Příklad	Výsledek
<code>&</code>	Adresa proměnné	<code>&x</code>	Ukazatel (pointer) na <code>x</code>
<code>*</code>	Nepřímá adresa	<code>*p</code>	Proměnná (nebo funkce) adresovaná pointerem <code>p</code>
<code>[]</code>	Prvek pole	<code>x[i]</code>	<code>*(x+i)</code> – prvek pole <code>x</code> s indexem <code>i</code>
<code>.</code>	Prvek struct/union	<code>s.x</code>	Prvek <code>x</code> struktury <code>s</code>
<code>-></code>	Prvek struct/union	<code>p->x</code>	Prvek struktury adresovaný ukazatelem <code>p</code>

*Operandem operátoru & nesmí být bitové pole a proměnná typu register. Operátor nepřímé adresy * umožňuje přístup na proměnné přes ukazatel.*

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 46 / 57

Výrazy a operátory Přirazení

Ostatní operátory

- Operandem `sizeof()` může být jméno typu nebo výraz.

<code>()</code>	Volání funkce	<code>f(x)</code>	Volání funkce <code>f</code> s argumentem <code>x</code>
<code>(type)</code>	Přetypování (cast)	<code>(int)x</code>	Změna typu <code>x</code> na <code>int</code>
<code>sizeof</code>	Velikost prvku	<code>sizeof(x)</code>	Velikost <code>x</code> v bajtech
<code>?:</code>	Podmíněný příkaz	<code>x ? y : z</code>	Proveď <code>y</code> pokud <code>x != 0</code> jinak <code>z</code>
<code>,</code>	Postupné vyhodnocení	<code>x, y</code>	Vyhodnotí <code>x</code> pak <code>y</code> , výsledek operátoru je výsledek posledního výrazu
- Operandem operátoru `sizeof()` může být jméno typu nebo výraz.


```
int a = 10;
printf("%lu %lu\n", sizeof(a), sizeof(a + 1.0));
```
- Příklad použití operátoru čárka.


```
for (c = 1, i = 0; i < 3; ++i, c += 2) {
    printf("i: %d c: %d\n", i, c);
}
```

[lec03/sizeof.c](#)

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 47 / 57

Výrazy a operátory Přirazení

Operátor přetypování

- Změna typu za běhu programu se nazývá přetypování.
- Explicitní přetypování (cast) zapisuje programátor uvedením typu v kulatých závorkách, např.


```
int i;
float f = (float)i;
```
- Implicitní přetypování provádí překladač automaticky při překladu.
- Pokud nový typ může reprezentovat původní hodnotu, přetypování ji vždy zachová.
- Operandy typů `char`, `unsigned char`, `short`, `unsigned short`, případně bitová pole, mohou být použity tam, kde je povolen typ `int` nebo `unsigned int`.
 - Operandy jsou automaticky přetypovány na `int` nebo `unsigned int`.

C očekává hodnoty alespoň typu int.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 48 / 57

Výrazy a operátory Přirazení

Asociativita a priorita operátorů

- Binární operace op na množině `S` je **asociativní**, jestliže platí $(x \text{ op } y) \text{ op } z = x \text{ op } (y \text{ op } z)$, pro každé $x, y, z \in S$.
- U **neasociativních operací** je nutné řešit v jakém pořadí jsou operace implicitně provedeny.
 - Asociativní zleva – operace jsou seskupeny zleva.

Např. výraz $10 - 5 - 3$ je vyhodnocen jako $(10 - 5) - 3$
 - Asociativní zprava – operace jsou seskupeny zprava.

Např. $3 + 5^2$ je 28 nebo $3 \cdot 5^2$ je 75 vs. $(3 \cdot 5)^2$ je 225
- Přirazení je asociativní zprava, např.

$y = y + 8.$
Vyhodnotí se nejprve celá pravá strana operátoru `+=`, která se následně přiřadí do proměnné na straně levé.
- Priorita binárních operací vyjadřuje v algebře pořadí, v jakém jsou binární operace prováděny.
- Pořadí provedení operací lze definovat důsledným **závorkováním**.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 49 / 57

Výrazy a operátory Přirazení

Přirazení

- Nastavení hodnoty proměnné. *Uložení definované hodnoty na místo v paměti.*
- Tvar přiřazovacího operátoru. $\langle \text{proměnná} \rangle = \langle \text{výraz} \rangle$

Výraz je literál, proměnná, volání funkce, ...

Přirazení je výraz, který můžeme použít v jiném výrazu, např. `a = b = c = 10;` *Je to výraz v příkazu přiřazení.*

- C je staticky typovaný jazyk.
 - Proměnné lze přiřadit hodnotu výrazu pouze identického typu. *Jinak je nutné provést typovou konverzi.*
 - Příklad implicitní konverze při přiřazení.

```
int i = 320.4; // implicit conversion from 'double' to 'int' changes value from 320.4 to 320 [-Wliteral-conversion]
```

```
char c = i; // implicit truncation 320 -> 64
```

- C je typově bezpečné v omezeném kontextu kompilace, např. na `printf("%d\n", 10.1);` kompilátor upozorní na chybu. **Obecně není typově bezpečné.** *Za běhu programu může dojít například k zápisu mimo vyhrazenou paměť a tím může dojít k nedefinovanému chování.*

„syntactic sugar“

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 51 / 57

Výrazy a operátory Přirazení

Zkrácený zápis přiřazení

- Zápis $\langle \text{proměnná} \rangle = \langle \text{proměnná} \rangle \langle \text{operátor} \rangle \langle \text{výraz} \rangle$
- Lze zapsat zkráceně $\langle \text{proměnná} \rangle \langle \text{operátor} \rangle = \langle \text{výraz} \rangle.$

Příklad

<code>int i = 10;</code>	<code>int i = 10;</code>	
<code>double j = 12.6;</code>	<code>double j = 12.6;</code>	
<code>i = i + 1;</code>	<code>i += 1;</code>	
<code>j = j / 0.2;</code>	<code>j /= 0.2;</code>	

- Přiřazení je výraz

```
int x, y;

x = 6;
y = x = x + 6;
```

„syntactic sugar“

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 52 / 57

Výrazy a operátory Přirazení

Výraz a příkaz

- Příkaz provádí akci a je zakončen středníkem.


```
robot_heading = -10.23;
robot_heading = fabs(robot_heading);
printf("Robot heading: %f\n", robot_heading);
```
- Výraz má určený **typ a hodnotu.**

<code>23</code>	typ <code>int</code> , hodnota 23
<code>14+16/2</code>	typ <code>int</code> , hodnota 22
<code>y=8</code>	typ <code>int</code> , hodnota 8
- Přiřazení je výraz a jeho hodnotou je hodnota přiřazená levé straně.
- Z výrazu se stává příkaz, pokud je ukončen středníkem.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 53 / 57

Část III

Zadání 2. domácího úkolu (HW2)

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 54 / 57

Zadání 2. domácího úkolu HW2

Téma: Kreslení (ASCII art) *Povinné zadání: 3b; Volitelné zadání: není; Bonusové zadání: není*

- Motivace:** Zábavným a tvůrčím způsobem získat praktickou zkušenost s cykly a jejich parametrizací na základě uživatelského vstupu.
- Cíl:** Osvojit si použití cyklů a vnořených cyklů.
- Zadání:** <https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw2>
 - Načtení parametrizace pro vykreslení šroubovice s využitím vybraných ASCII znaků. https://en.wikipedia.org/wiki/ASCII_art
 - Ošetření vstupních hodnot.
- Termín odevzdání:** 23.03.2024, 23:59:59 PDT. *PDT – Pacific Daylight Time*

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 55 / 57

Shrnutí přednášky

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 56 / 57

Diskutovaná témata

Diskutovaná témata

- Řídící struktury - přepínač, cykly, vnořené cykly, `break` a `continue`
- Konečnost cyklů
- Kódovací konvence
- Výrazy - unární, binární a ternární
- Přehled operátorů a jejich priorit
- Přiřazení a zkrácený způsob zápisu
 - Příkazy a nedefinované chování
- Příště:** Pole, ukazatel, textový řetězec, vstup a výstup programu.

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 57 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Část V

Appendix

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 58 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Kódovací příklad – Tisk hodnot v šestnáctkové soustavě

- Reprezentace `float` hodnot. `#include <stdio.h>`
 - Hodnota 85.125 je `0x42aa4000`.
 - Hodnota 0.1 je sice `0x3dcccc`, ale je kódována `0x3dccccd`. *Protože chyba je absolutně menší.*
- Implementujeme funkci pro tisk paměťové reprezentace hodnoty typu `float` jako posloupnosti hodnot bajtů v šestnáctkové soustavě. `void print_float_hex(float v);`
- Přístup k `float` jako posloupnosti bajtů a tisk hex hodnot `"%02x"` funkcí `printf()`. `int main(void)`

```
{
    print_float_hex(85.125);
    print_float_hex(0.1);
    return 0;
}
```
- Adresním operátorem `&` získáme adresu proměnné. `void print_float_hex(float v)`
- Přetypujeme adresu jako ukazatel na hodnotu `char`. `{`
- Použijeme nepřímý adresní operátor `*` k přístupu k hodnotě na adrese uložené v ukazateli. `{`

```
...
}
```

Jan Faigl, 2024 BAB36PRGA – Přednáška 03: Řídící struktury, výrazy a funkce 60 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Kódovací příklad – Tisk hodnot v šestnáctkové soustavě 1/3

- Získáme adresu proměnné `float v` operátorem `&v`.
- K hodnotám na adrese `&v` budeme přistupovat jako k bajtům, proto přetypujeme adresu na ukazatel (adresu) na hodnoty typu `char`.
`unsigned char *p = (unsigned char*)&v;`
- Hodnotu uloženou na adrese `p` získáme operátorem nepřímého adresování `*p`.
Protože se jedná o ukazatel na char, probíhá inkrementace o `sizeof(char)`, tj. o 1 (ukazatelová aritmetika).
- Adresu následujícího bajtu za adresu uloženou v `p` získáme `p = p + 1;`.
- Vytiskněné hodnoty jsou v opačném než očekávaném pořadí `0x42aa4000` a `0x3dcccccd`.

```
int main(void)
{
    print_float_hex(85.125);
    print_float_hex(0.1);
    ...
    void print_float_hex(float v)
    {
        unsigned char *p = (unsigned char*)&v;
        printf("Value %13.10f is 0x", v);
        for (int i = 0; i < 4; ++i, p = p + 1) {
            printf("%02x", *p); // or use p[i]
        }
        putchar('\n');
    }
}

$ clang floats.c -o floats && ./floats
Value 85.1250000000 is 0x0040aa42
Value 0.1000000015 is 0xcdcccc3d
```

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 61 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Kódovací příklad – Tisk hodnot v šestnáctkové soustavě 2/3

- Očekávaná reprezentace v šestnáctkové soustavě je pro `85.125` výstup `0x42aa4000` a pro `0.1` výstup `0x3dcccccd`. Namísto toho dostáváme `0x0040aa42` a `0xcdcccc3d`.
- Výstup je závislý na reprezentaci více bajtových hodnot v paměti. Pro architekturu (amd64) je to tzv. little endian.
<https://en.wikipedia.org/wiki/Endianness>
- Proto potřebujeme detekovat, jak jsou hodnoty uloženy, například funkcí `_Bool is_big_endian(void);`
- a případně vytiskneme hodnoty v opačném pořadí.

```
void print_float_hex(float v)
{
    const _Bool big_endian = is_big_endian();
    // cast pointer to float to pointer to char
    unsigned char *p = (unsigned char*)&v
    + (big_endian ? 0 : 3);
    printf("Value %13.10f is 0x", v);
    for (int i = 0; i < 4; ++i) {
        printf("%02x",
            *(big_endian ? p++ : p--));
    }
    printf("\n");
}

$ clang floats.c -o floats && ./floats
Value 85.1250000000 is 0x42aa4000
Value 0.1000000015 is 0x3dcccccd
```

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 62 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Kódovací příklad – Tisk hodnot v šestnáctkové soustavě 3/3

- Detekce uložení můžete být založena na různých principech.
- Intuitivně můžeme uložit definovanou hodnotu, která má pouze jeden bajt nenulový a ostatní nulové.
- Využijeme složeného typu `union`, ve kterém položky sdílejí paměť a umožňují nám tak různý pohled na konkrétní block paměti.

```
#include <stdint.h>
_Bool is_big_endian(void)
{
    union {
        uint32_t i;
        char c[4];
    } e = { 0x01000000 };
    return e.c[0];
}
```

- Definujeme celočíselnou proměnnou o čtyřech bajtech, např., `uint32_t` z knihovny `stdint.h`.
- Nastavíme hodnotu na `0x01 00 00 00`.
- Otestujeme první bajt paměťové reprezentace.

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 63 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Nedefinované chování

- Dle standardu C mohou některé příkazy (výrazy) způsobit **nedefinované chování**.
 - `c = (b = a + 2) - (a - 1);`
 - `j = i * i++;`
- Program se může chovat rozdílně podle použitého kompilátoru, případně nemusí jít zkompileovat, spustit, nebo dokonce padat a chovat se neobvykle či produkovat nesmyslné výsledky.
- To se může například také stát v případě, že nejsou proměnné inicializovány.

Vyhýbejte se příkazům (výrazům), které mohou vést na nedefinované chování!

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 65 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Příklad nedefinovaného chování

- Standard C nepředpisuje chování při přetečení celého čísla (`signed`)
 - V případě doplňkového kódu může být např. hodnota výrazu `127 + 1` typu `char` rovna `-128`, viz `lec03/demo-loop_byte.c`.
 - Reprezentace celých čísel však může být realizována jinak dle architektury např. přímým kódem nebo inverzním kódem.
- Zajištění předepsaného chování tak může být výpočetně komplikované, proto standard nedefinuje chování při přetečení.
- Chování programu není definované a závisí na kompilátoru**, např. překladače `clang` a `gcc` bez/s optimalizacími `-O2`.
 - `for (int i = 2147483640; i >= 0; ++i) { printf("%i %x\n", i, i); }`
 Bez optimalizací program vypíše 8 řádků, pro `-O2` program zkompileovaný `clang` vypíše 9 řádků, `gcc` program skončí v nekonečné smyčce.
 - `for (int i = 2147483640; i >= 0; i += 4) { printf("%i %x\n", i, i); }`
 Program zkompileovaný `gcc` s `-O2` po spuštění (může) padat(at).

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 66 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Compiler Explorer – Analýza optimalizovaného kódu

- Vliv optimalizace `-O2` na výsledný kód, který obsahuje nedefinované chování, přetečení celého čísla.

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 67 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Přehled operátorů a jejich priorit 1/3

Priorita	Operátor	Asociativita	Operace
1	<code>++</code> <code>--</code> <code>()</code> <code>[]</code> <code>.</code> <code>-></code>	P/L L→P	pre/post inkrementace pre/post dekrementace volání metody indexace do pole přístup na položky struktury/unionu přístup na položky přes ukazatel
2	<code>! ~</code> <code>+</code> <code>+</code> <code>()</code> <code>*</code> <code>&</code> <code>sizeof</code>	P→L	logická a bitová negace unární plus (minus) přetypování nepřímé adresování (dereference) adresa (reference) velikost

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 69 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Přehled operátorů a jejich priorit 2/3

Priorita	Operátor	Asociativita	Operace
3	<code>*</code> , <code>/</code> , <code>%</code>	L→R	násobení, dělení, zbytek
4	<code>+</code> , <code>-</code>		scítání, odečítání
5	<code>>></code> , <code><<</code>		bitový posun vlevo, vpravo
6	<code><</code> , <code>></code> , <code><=</code> , <code>>=</code>		porovnání
7	<code>==</code> , <code>!=</code>		rovnno, nerovno
8	<code>&</code>		bitový AND
9	<code>^</code>		bitový XOR
10	<code>^</code>		bitový OR
1	<code>&&</code>		logický AND
12	<code> </code>		logický OR

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 70 / 57

Kódovací příklad Nedefinované chování Přehled operátorů a jejich priorit

Přehled operátorů a jejich priorit 3/3

Priorita	Operátor	Asociativita	Operace
13	<code>?:</code>	P→L	ternární operátor
14	<code>=</code> <code>+=</code> , <code>-=</code> <code>*=</code> , <code>/=</code> , <code>%=</code> <code><<=</code> , <code>>>=</code> <code>&=</code> , <code>^=</code> , <code> =</code>	P→L	přičtení přičtení součtu, rozdíl přičtení součinu, podílu a zbytku přičtení bitového posunu vlevo, vpravo přičtení bitového AND, XOR, OR
15	<code>,</code>	L→P	operátor čárka

http://en.cppreference.com/w/c/language/operator_precedence

Jan Faigl, 2024 BAB36PRGA – Prednáška 03: Řídící struktury, výrazy a funkce 71 / 57