



PROGRAMOVACIE JAZYKY PRE VSTAVANÉ SYSTÉMY

Štandardné dátové typy, pamäťové triedy, príkazy

OTÁZKY Z MINULEJ PREDNÁŠKY

- Čo je to deklarácia a definícia?
- Čo rozumieme pod pojmom oblasť viditeľnosti identifikátora?
- Aké štandardné dátové typy pozná jazyk C?
- Čo je to l-hodnota?
- Aký je rozdiel medzi konštantou a literálom?
- Čo sú to „usual arithmetic conversions“?
- Určte, akého typu budú nasledujúce výrazy:
 - $10u - 50$
 - $0UL - 1LL$
 - $!(10 < 1 \ \&\& \ !(10.5) == 1)$
- Určte výsledok a typ nasledujúceho výrazu:
 - $7f + 5u / 3$



ŠTANDARDNÉ DÁTOVÉ TYPY (1)

○ Základné (aritmetické):

- znak (**char**)
- celočíselné znamienkové (signed char, short, **int**, long, **long long** (C99))
- celočíselné neznamienkové (_Bool (C99), unsigned char, unsigned short, unsigned int, unsigned long, unsigned long long (C99))
- s pohyblivou desatinnou čiarkou:
 - reálne: **float**, **double**, long double
 - komplexné (C99) (float _Complex, double _Complex, long double _Complex)
 - imaginárne (C99) (float _Imaginary, double _Imaginary, long double _Imaginary)



ŠTANDARDNÉ DÁTOVÉ TYPY (2)

- Vymenovaný typ (**enum**)
- Typ **void**
- Odvodené:
 - **pole**
 - štruktúra (**structure**)
 - zjednotenie (**union**)
 - **ukazovateľ**
 - atomické typy (C11) (**_Atomic**) – aplikovateľné na ľubovoľný dátový typ okrem poľa



DÁTOVÝ TYP ZNAK (CHAR)

- **char** – údajový typ reprezentujúci 1 znak zo základnej znakovkej sady.
- Ak je v objekte typu char uložený znak zo základnej znakovkej sady (basic execution character set), hodnota je vždy nezáporná.
- Ak je v objekte typu char uložený iný znak, hodnota je implementačne závislá, ale musí byť v rozsahu hodnôt, ktoré je možné týmto typom reprezentovať.
- Typ char je buď signed char, alebo unsigned char, je to implementačne závislé (char, signed char a unsigned char sú 3 odlišné dátové typy).
- Veľkosť typu char je vždy **1 bajt**.



CELOČÍSELNÉ DÁTOVÉ TYPY (1)

Dátový typ	Veľkosť (štandard)	Windows/ Unix 32 bitový	Windows 64 bitový	Unix 64 bitový
signed char	1 bajt	8 bitov	8 bitov	8 bitov
short	aspoň 16 bitov	16 bitov	16 bitov	16 bitov
int	aspoň 16 bitov	32 bitov	32 bitov	32 bitov
long	aspoň 32 bitov	32 bitov	32 bitov	64 bitov
long long	aspoň 64 bitov	64 bitov	64 bitov	64 bitov

$1 == \text{sizeof}(\text{char}) \leq \text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long}) \leq \text{sizeof}(\text{long long})$

- **sizeof** – operátor, ktorý vracia veľkosť objektovej reprezentácie dátového typu alebo výrazu v **bajtoch**



CELOČÍSELNÉ DÁTOVÉ TYPY (2)

- Znamienkové – štandard nedefinuje spôsob reprezentácie

Celočíselný dátový typ		Minimum	Maximum
neznamienkový		0	$2^n - 1$
znamienkový	jednotkový doplnok (inverzný kód)	$-2^{n-1} + 1$	$2^{n-1} - 1$
	dvojkový doplnok (doplnkový kód)	-2^{n-1}	$2^{n-1} - 1$

* n – veľkosť v bitoch

Celé číslo	Inverzný kód	Doplnkový kód
127	0111 1111	0111 1111
1	0000 0001	0000 0001
0	0000 0000	0000 0000
(-0)	1111 1111	0000 0000
-1	1111 1110	1111 1111
-127	1000 0000	1000 0001



REÁLNE DÁTOVÉ TYPY (1)

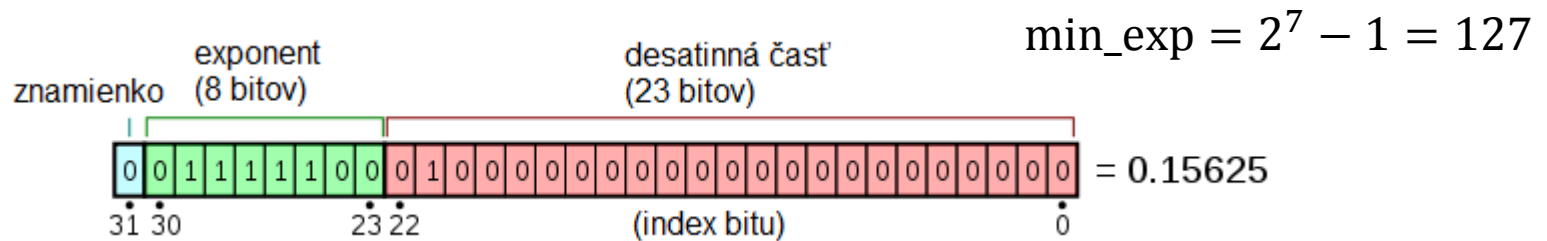
- **float** – reálne čísla s jednoduchou presnosťou (napr. IEEE-754 32 bit floating point type)
- **double** – podporuje aspoň takú presnosť ako float, množina hodnôt tohto typu je nadmnožinou hodnôt typu float (napr. IEEE-754 64 bit floating point type)
- **long double** – podporuje aspoň takú presnosť ako double, množina hodnôt tohto typu je nadmnožinou hodnôt typu double (napr. IEEE-754 extended floating point type, double)



REÁLNE DÁTOVÉ TYPY (2)

$$(-1)^{\text{sgn}} \left(1 + \sum_{i=1}^{p-1} \text{bit}_{p-1-i} 2^{-i} \right) 2^{\text{exp} - \text{min_exp}}$$

IEEE-754 32 bit floating point type



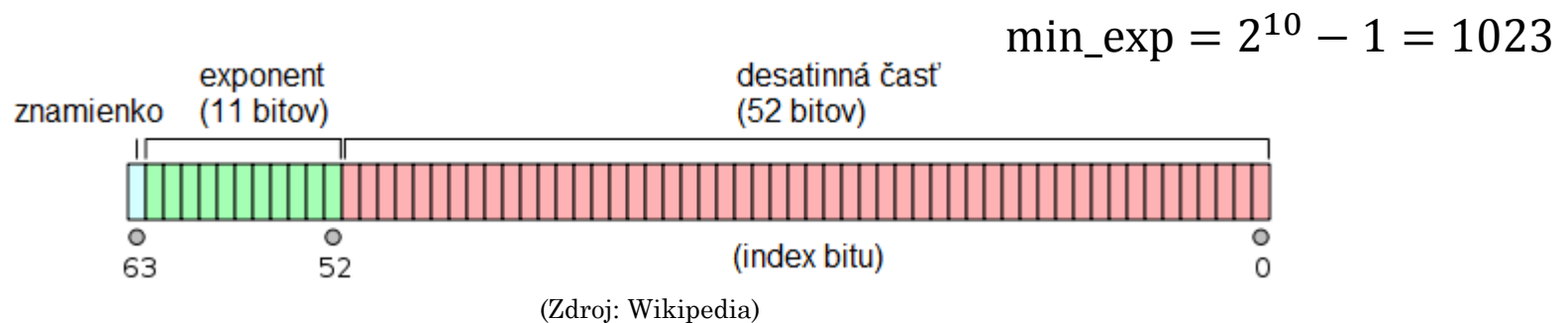
(Zdroj: Wikipedia)

$$(-1)^{b_{31}} \times (1, b_{22}b_{21} \dots b_0)_2 \times 2^{(b_{30}b_{29} \dots b_{23})_2 - 127}$$

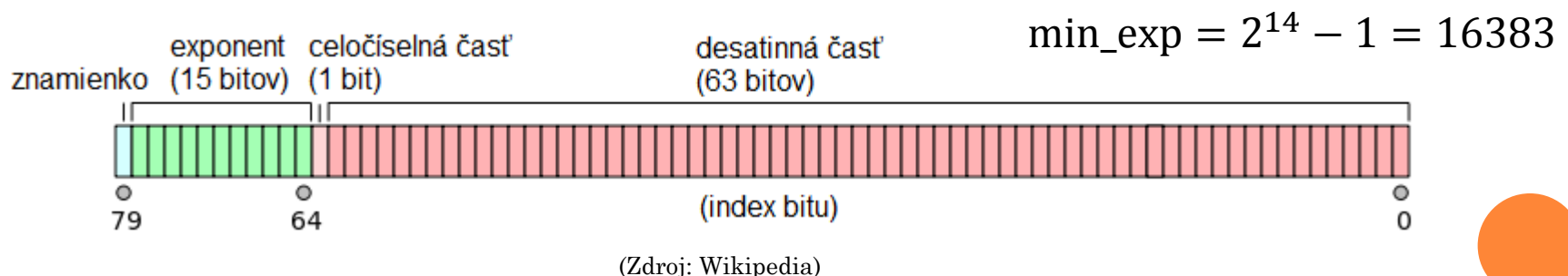


REÁLNE DÁTOVÉ TYPY (3)

- IEEE-754 64 bit floating point type



- IEEE-754 extended floating point type



KOMPLEXNÉ DÁTOVÉ TYPY (C99)

- **float _Complex, double _Complex, long double _Complex**
- Dátový typ _Complex je implementovaný ako dvojprvkové pole elementov typu float, double alebo long double.
- **float _Imaginary, double _Imaginary, long double _Imaginary**
- Dátový typ _Imaginary je implementovaný ako príslušný reálny typ.
- S dátovými typmi _Complex a _Imaginary je možné použiť nasledujúce operátory:
 - +, -, *, / (nepodporujú %, ++, --)
 - ==, != (nepodporujú <, <=, >, >=)
 - &&, ||, !



LOGICKÝ TYP

- Jazyk C bol pôvodne navrhnutý bez typu Boolean. Logické hodnoty sú reprezentované (**celými číslami** (najčastejšie typ int):
 - 0 – false
 - číslo rôzne od 0 – true
- V C99 bol pridaný dátový typ **_Bool**, ktorý má iba dve možné hodnoty – **číslo 0** alebo **číslo 1**.
- Určte, akú hodnotu majú nasledujúce výrazy:
 - (int)0.5
 - (_Bool)0.5



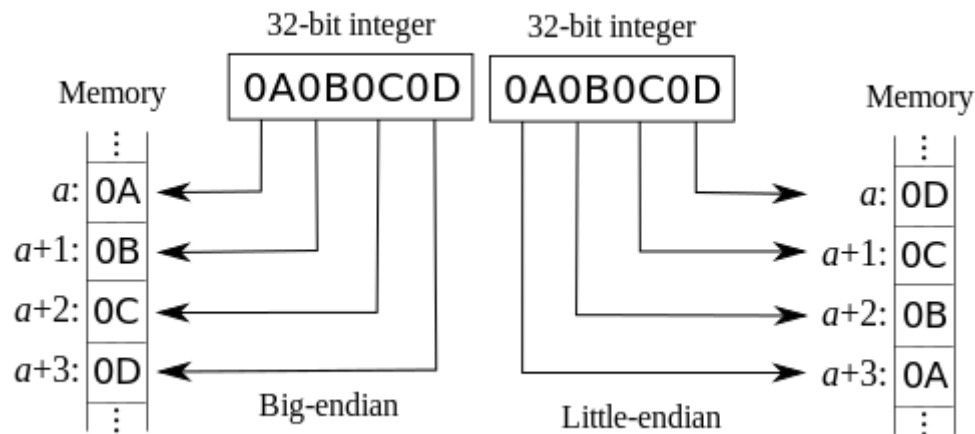
TYP VOID

- Neúplný typ.
- Nemá definovanú veľkosť (avšak v GCC platí, že `sizeof(void) = 1`).
- Nemôžeme definovať premennú typu void.
- Využitie:
 - generický smerník: **void***
 - funkčný prototyp:
 - indikátor, že funkcia je bez návratového typu: **void** fun(int a)
 - indikátor, že funkcia nemá žiadne parametre: int fun(**void**)



ENDIANITA

- **Endianita** – poradie, v akom sú bajty viacbajtového dátového typu uložené v pamäti alebo prenesené po sieti.
- **Big-endian** – Motorola 68000, Xilinx Microblaze, Power PC, **sieťové protokoly** IPv4, IPv6, TCP, UDP,...
- **Little-endian** – Intel **x86**, x86-64,...
- Middle-endian



(Zdroj: Wikipedia)

ŠTANDARDNÁ KNIŽNICA JAZYKA C

○ Hlavičkové súbory:

- **limits.h** – popis rozsahov celočíselných dátových typov a typu char:
 - CHAR_BIT, CHAR_MIN, CHAR_MAX, INT_MIN, INT_MAX,...
- **float.h** – popis rozsahov reálnych dátových typov:
 - FLT_RADIX, FLT_MIN, FLT_MAX,...
- **stdbool.h** (C99) – užitočné makrá pre prácu s dátovým typom _Bool
 - bool (typ _Bool), true (číslo 1), false (číslo 0)
- **math.h** – funkcie pracujúce s reálnymi číslami (sin, pow, log,...)
- **complex.h** (C99) – funkcie pracujúce s komplexnými číslami (sin, pow, log,...), užitočné makrá pre prácu s dátovými typmi _Complex a _Imaginary
 - imaginary, complex, _Imaginary_I, _Complex_I, I
- **stdlib.h**
- **stdio.h** – funkcie pracujúce so vstupom/výstupom (scanf, printf, fgets,...)
- ...



PREMENNÉ

- Premenná asociuje miesto v pamäti s identifikátorom.
- Pomocou premenných sa odvolávame na príslušné miesto v pamäti, kde je uložená hodnota daného typu.
- Adresu, na ktorej sa nachádza daná premenná, získame pomocou operátora **&** (address of).
- Môže existovať viacero **deklarácií** premennej (kľúčové slovo **extern**):
 - `extern <typ> <zoznam deklarátorov>;`
 - `extern int a, b;`
- Premenná môže byť **definovaná** maximálne jeden raz:
 - `<typ> <zoznam deklarátorov>;`
 - `int a;`
 - `int b = 10, c = 20;`
 - `extern int d = 30;` (len v prípade globálnych premenných; nepoužívať – zhoršuje čitateľnosť kódu)



TYPY PREMENNÝCH

- **Globálne** – platné (viditeľné) od miesta deklarácie do konca súboru.
- **Lokálne** – platné od miesta deklarácie do konca bloku, v ktorom boli deklarované.
- **Formálne parametre funkcie** – platné v hlavičke funkcie od miesta definície a v celom bloku funkcie.

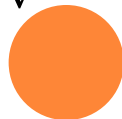


UKÁŽKA ZDROJOVÉHO KÓDU

```
1  #include <stdio.h>
2
3  extern int globA;
4
5  static void ukazkovaFunkcia() {
6      int i = 0;
7      static int j = 5;
8
9      printf("ukazkovaFunkcia\n");
10     printf("i = %d, j = %d\n", i, j);
11     printf("globA = %d\n", globA);
12     i++;
13     j++;
14 }
15
16 int globA;
17
18 int main(int argc, char** argv) {
19     printf("main\n");
20     printf("globA = %d\n", globA);
21     ukazkovaFunkcia();
22     globA++;
23     ukazkovaFunkcia();
24
25     return 0;
26 }
```



PAMÄŤOVÉ TRIEDY

- Určujú, v ktorej časti pamäti bude premenná uložená (teda aj jej životnosť) a kde všade bude viditeľná.
 - Jazyk C rozlišuje 5 pamäťových tried:
 - **auto** – implicitná trieda pre lokálne premenné. Premenné tejto triedy sú uchovávané na zásobníku, pričom existujú od miesta vstupu do bloku a zanikajú po opustení bloku, v ktorom boli definované.
 - **extern** – implicitná trieda pre globálne premenné. Premenné tejto triedy sú uchovávané v dátovom segmente, pričom existujú od okamžiku spustenia programu až do jeho ukončenia. Globálne premenné tejto triedy sú prístupné aj mimo súboru, v ktorom boli definované (t.j. v celom programe).
- 

PAMÄŤOVÉ TRIEDY

- **static** – premenné tejto triedy sú uchovávané v dátovom segmente, pričom existujú od okamžiku spustenia programu až do jeho ukončenia. Lokálna premenná tejto triedy sa správa ako súkromná globálna premenná bloku, v ktorom je definovaná. Globálna premenná tejto triedy je prístupná len v súbore, v ktorom bola definovaná.
- **register** - indikuje kompilátoru, že daná premenná sa bude intenzívne používať a mala by byť dostupná čo najrýchlejšie. Nie je možné získať adresu premennej triedy register. Tento modifikátor je možné použiť na formálne parametre funkcií a ich lokálne premenné.
- **_Thread_local** (C11)
- Modifikátory **extern** a **static** je možné použiť aj s funkciami. Vtedy určujú, či je funkcia viditeľná v celej aplikácii (**extern**) alebo len v danom súbore (**static**). Implicitný modifikátor je **extern**.



ORGANIZÁCIA PAMÄŤOVÉHO PRIESTORU

PROCESU

- **Kódový segment** – inštrukcie programu, len na čítanie.
- **Inicializovaný dátový segment** – globálne a statické premenné inicializované v kóde.
- **BSS** (z angl. „block started by symbol“) – globálne a statické premenné, ktoré nie sú explicitne inicializované v kóde; pri zavedení programu sú premenné v tomto segmente inicializované na 0.
- **Zásobník** („stack“) – automatické premenné (lokálne premenné, ktoré nie sú statické), argumenty funkcie,...
- **Halda** („heap“) – dynamické premenné spravované funkciami malloc, calloc, realloc a free.



(Zdroj: Wikipedia)

PRÍKAZ

- Pokyn na vykonanie nejakej činnosti.
- Jazyk C pozná 5 typov príkazov:
 - **(výrazový) príkaz:**
`n = faktorial(5) + x;`
 - **prázdny príkaz:**
`;`
 - **zložený príkaz (blok):**

```
{  
    x = 3 + y;  
    y++;  
}
```
 - **príkazy vetvenia** (if-else, switch)
 - **príkazy cyklov** (for, while, do-while)
 - **príkazy skoku** (break, continue, return, goto)
- Za každým príkazom sa nachádza „sequence point“.
- Hociktorý príkaz môže byť označený 1 alebo viacerými návestiami (popiskami, „label“):
`fakt: n = faktorial(5) + x;`



VETVENIE IF-ELSE

- Neúplné vetvenie:

if (výraz) príkaz_true

- Úplné vetvenie

if (výraz) príkaz_true else príkaz_false

- výraz – akýkoľvek výraz aritmetického alebo smerníkového typu
- príkaz_true – akýkoľvek typ príkazu; vykoná sa, ak má výraz hodnotu rôznu od 0
- príkaz_false – akýkoľvek typ príkazu; vykoná sa, ak má výraz hodnotu 0

```
int c = 0;
if (c = getchar() - '0')
    puts("Znak je rozny od 0.\n");
else
    puts("Znak sa rovna 0.\n");
```



TERNÁRNY OPERÁTOR

výraz_pod ? výraz_true : výraz_false

- výraz_pod – akýkoľvek výraz aritmetického alebo smerníkového typu
- výraz_true – výraz, ktorý sa vyhodnotí, ak má výraz_pod hodnotu rôznu od 0
- výraz_false – výraz, ktorý sa vyhodnotí, ak má výraz_pod hodnotu 0
- výraz_true aj výraz_false musia byť výrazy aritmetického alebo smerníkového typu, alebo typu void, alebo musia predstavovať rovnaký typ štruktúry alebo zjednotenia („union“)
- po vyhodnotení výraz_pod je „sequence point“
- operátor má typ, ktorý je spoločný pre výraz_true a výraz_false
- výsledný výraz nie je l-hodnota

```
getchar() - '0' ? puts("Znak je rozny od 0.\n")  
               : puts("Znak sa rovna 0.\n");
```

- Určte, či je nasledujúci výraz korektný:
 - `a < b ? a : b = 5 /*a, b sú typu int*/`



PREPÍNAČ (SWITCH)

```
switch (výraz) {  
    case konštantný_výraz_1: príkazy_1 break;  
    case konštantný_výraz_2 : príkazy_2 break;  
    ...  
    case konštantný_výraz_N : príkazy_N break;  
    default : príkaz_default  
}
```

- výraz – akýkoľvek výraz celočíselného typu (char, _Bool, znamienkové/neznamienkové celočíselné typy, enum)
- konštantný_výraz_1, konštantný_výraz_1,... – akýkoľvek výraz celočíselného typu
- príkazy_1, príkazy_2,... – akékoľvek typy príkazov



CYKLUS WHILE

while (výraz) príkaz

- výraz – akýkoľvek výraz aritmetického alebo smerníkového typu; výraz sa vyhodnocuje pred každou iteráciou
- príkaz – akýkoľvek typ príkazu; príkaz sa opakuje pokiaľ má výraz hodnotu rôznu od 0

```
int i = 1;
int sucet = 0;
while (i <= 10)
{
    sucet += i;
    i++;
}
```



CYKLUS DO-WHILE

do príkaz while (výraz);

- výraz – akýkoľvek výraz aritmetického alebo smerníkového typu; výraz sa vyhodnocuje na konci každej iterácie
- príkaz – akýkoľvek typ príkazu; príkaz sa vykoná vždy aspoň raz, pričom sa opakuje pokiaľ má výraz hodnotu rôznu od 0

```
int i = 1;
int sucet = 0;
do
{
    sucet += i;
    i++;
} while (i <= 10)
```



CYKLUS FOR

for ([init]; [výraz_pod]; [výraz_iter]) príkaz

- `init` – buď akýkoľvek výraz, ktorý je vyhodnotený len pri vstupe do cyklu, alebo deklarácia (C99)
- `výraz_pod` – akýkoľvek výraz aritmetického alebo smerníkového typu; `výraz_pod` sa vyhodnocuje pred každou iteráciou
- `výraz_iter` – akýkoľvek výraz; `výraz_iter` sa vykonáva na konci každej iterácie
- `príkaz` – akýkoľvek typ príkazu; príkaz sa opakuje pokiaľ má `výraz_pod` hodnotu rôznu od 0

```
int sucet = 0;
for (int i = 1; i <= 10; i++)
{
    sucet += i;
}
```

- `for (; ;)` – nekonečný cyklus



PRÍKAZY BREAK, CONTINUE

- **break** – spôsobí okamžité ukončenie príkazu switch alebo cyklu (while, do-while, for)
- **continue** – spôsobí okamžitý skok za príkaz, ktorý sa opakuje v rámci cyklu (while, do-while, for)

```
int sucet = 0;
for (int i = 1; ; i++)
{
    if (i > 10)
        break;
    if (i % 2)
        continue;
    sucet += i;
}
```



OPERÁTOR , (ZABUDNUTIA)

- Operátor čiarka sa používa, ak na mieste, kde sa očakáva jeden výraz, chceme zapísať a vyhodnotiť viac výrazov.
- Operátor má najmenšiu prioritu spomedzi všetkých operátorov.
- Pre výraz **A**, **B** platí:
 - A a B môžu byť výrazy ľubovoľného typu
 - výraz **A**, **B** vracia hodnotu B ako r-hodnotu, a preto sa jeho typ zhoduje s typom B
 - vždy sa najskôr vyhodnotí výraz A a následne výraz B
 - za výrazom A sa nachádza „sequence point“
- Typické využitie – cykly:
 - `for(int i=0, j=0; i < 100; i++, j--)`
- Určte, čo bude v premennej c:

<code>int a = 1, b = 2, c;</code>	<code>int a = 1, b = 2, c;</code>
<code>c = (a, b++, b);</code>	<code>c = (a, b, b++);</code>
- Prepíšte predchádzajúce cykly s využitím operátora čiarka.



VYHODNOCOVANIE VÝRAZOV A „SEQUENCE POINT“

- Jazyk C vo všeobecnosti nedefinuje, v akom poradí sa vyhodnocujú jednotlivé operandy vo výraze; napr.:
 - `fun1() + fun2()`
 - `fun1() + fun2() + fun3()`
 - `fun1() + (fun2() + fun3())`
 - ani v jednom nevieme povedať, či sa najskôr zavolá funkcia `f1()` alebo `f2()`, alebo `f3()`
- Existujú však miesta, pre ktoré je garantované, že všetky „side effects“ vyplývajúce z predchádzajúcich výpočtov boli aplikované – tzv. „sequence points“.
- „Sequence points“ sa nachádzajú napr.:
 - medzi vyhodnotením argumentov funkcie a volaním samotnej funkcie
 - po vyhodnotení 1. operandu v operátoroch `&&`, `||`, `,`
 - po vyhodnotení 1. operandu v ternárnom operátore
 - za každým príkazom
 - ...

