

Informatika 3

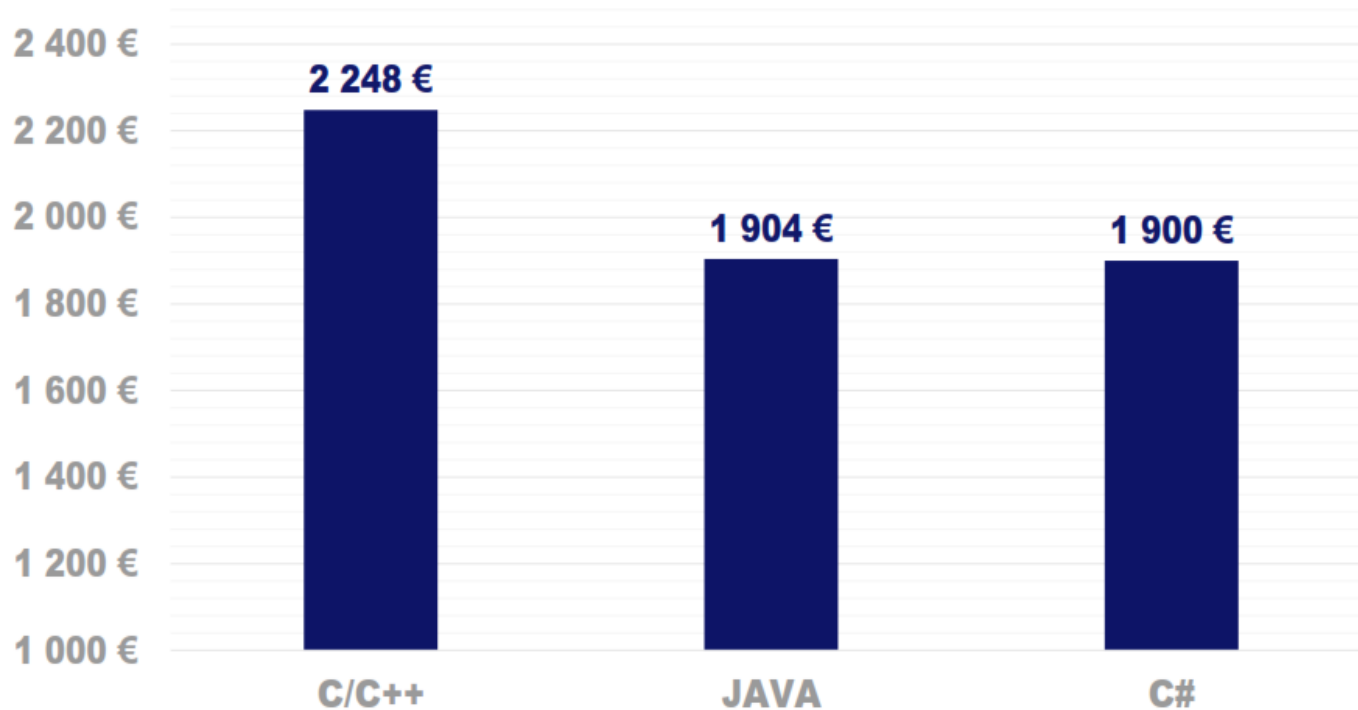
2.

Prečo C/C++ !

HayGroup®

Celkový príjem programátorov

Hay Group Prieskum Odmeňovania (Vzorka 1300 jednotlivcov z 24 firiem.)



Deklarácia

Oznamuje kompilátoru:

„*Táto funkcia alebo premenná niekde existuje a bude vypadat' takto.*“



```
int  func(int x, int y);    // deklarácie funkcie
extern int  a;              // deklarácia premennej
```

- Môže byť viacnásobná (ak je zhodná)
- Väčšinou sa umiestňuje do hlavičkového (včleňovaného) súboru
- Deklarácie funkcie = **prototyp funkcie**
 - Prototyp funkcie musíme umiestniť pred prvé použitie funkcie
 - Prototyp funkcie napíšeme explicitne do zdrojového kódu sami alebo
 - Včleníme do zdrojového kódu hlavičkový súbor, ktorý tento prototyp obsahuje.

Definícia

Prikazuje kompilátoru:

„*Vytvor túto premennú na tomto mieste*“

alebo

„*Vytvor túto funkciu na tomto mieste*“.



```
int a;                // definícia premennej
int func1(int,int)    // definícia funkcie
{
    // ... telo funkcie
}
```

- Alokuje pamäť pre identifikátor.
- Môže byť iba jedna (ODR – one definition rule)
- Umiestňuje sa do zdrojového súboru

- Flexibilné deklarácie (C++, C99)



Hlavička funkcie ako rozhranie

- rozhranie medzi **volanou** funkciou a **volajúcou** funkciou

návratový typ meno funkcie

int fun (long pocet)

zoznam parametrov (argumentov)



- **návratový typ** - popis informácie, ktorá ide z volanej funkcie do volajúcej funkcie
- **zoznam parametrov (argumentov)** - popis informácie, ktorá ide z volajúcej do funkcie volanej funkcie

Prototyp funkcie

Deklarácie funkcie = prototyp funkcie



- Prototyp funkcie musíme umiestniť pred prvé použitie funkcie
- Prototyp funkcie napíšeme explicitne do zdrojového kódu sami alebo
- Včleníme do zdrojového kódu hlavičkový súbor, ktorý tento prototyp obsahuje

Funkcie

S návratovou hodnotou - vytvára a vracia hodnotu

- môžeme ju priradiť do premennej alebo
- použiť v inom výraze

```
double sqrt(double) ;  
x = sqrt(6.25) ;
```

Bez návratovej hodnoty - niečo vykonáva, nevracia hodnotu

- procedúry alebo podprogramy

```
void fun(double) ;  
fun(10.3) ;
```

Knižničné funkcie

- uložené v knižničných súboroch
- automatické prehľadávanie knižníc a pripájanie knižnice
- explicitné prehľadávanie knižníc (-lm)
- štandardná knižnica C jazyka - viac ako 140 preddefinovaných funkcií

**Ak použitie štandardnej funkcie postačuje,
treba ju použiť a nevytvárať vlastnú.**



Funkcie definované programátorom

- prototyp funkcie – v hlavičkovom súbore
- definícia tela funkcie – v zdrojovom súbore

```
// mojafun.h - hlavičkový súbor
```

```
void mojaFun(int pocet);
```

```
// mojafun.c - zdrojový súbor
```

```
#include "mojaFun.h"
```



```
void mojaFun(int pocet)
{
    // ... kód
}
```


```
// hlavny.c- vstupný bod programu
```

```
#include "mojaFun.h"
```



```
int main(void)
{
    mojaFun(10);
}
```

Kľúčové slová

- slovník počítačového jazyka
- **nepoužívať na iné účely (ako identifikátory)** 

Konvencie pomenovania

- závisia od riešiteľského tímu
- mali by byť jednotné 

Premenné

Nástroj na identifikáciu dát



Program musí sledovať 3 vlastnosti:



- Kde je informácia uložená
- Akú hodnotu uchováva
- O aký druh informácie sa jedná

Názvy premenných

Názov musí byť zmysluplný !



Pravidlá tvorby:

1. V menách môžeme používať písmena abecedy, číslice a podtržítka (_).
2. Prvým znakom mena nesmie byť číslica. 
3. Malé a veľké písmena sa rozlišujú. 
4. Ako názov nemôžeme použiť kľúčové slovo jazyka C++.
5. Názvy, začínajúce podtržítkom alebo dvomi podtržítkami sú rezervované pre použitie kompilátorom a prostriedkami, ktoré používa.
6. C++ neohraničuje dĺžku názvu a všetky znaky mena sú významné.
(Avšak niektoré platformy môžu mať svoje vlastné limity. ANSI C99 garantuje len 63 znakov.)

Štandardné celočíselné typy

- **char** - má minimálnu šírku 8 bitov
- **short** - má minimálnu šírku 16 bitov
- **int** - je minimálne taký veľký ako short
- **long** - má minimálnu šírku 32 bitov a je minimálne taký ako int
- **long long** - má minimálnu šírku 64 bitov a je minimálne taký ako long
- **wchar_t** - široký znak – variabilná šírka

C++11

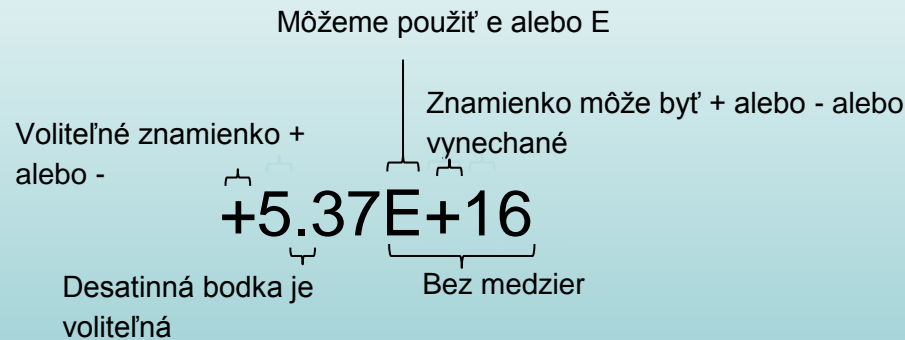
- **char16_t** - šírka 16 bitov
- **char32_t** - šírka 32 bitov

- Štandardne **signed**
- Modifikátor **unsigned**



Štandardné typy pohyblivej rádovej bodky

- **float** - 32 bitov
- **double** - 48 bitov a nie menej float
- **long double** - nie menej ako double (80, 96, 128)



- čísla medzi celými číslami
- omnoho väčší rozsah hodnôt
- strata presnosti



Číselné konštanty

Celočíselné


- celé číslo 1536
- celé čísla bez znamienka (54321u, 31U)
- hexadecimálne konštanty (**0x**31, **0X**1b2C)
- oktálové konštanty (**0**15, **0**324)
- Celé číslo typu long 1536**l** alebo 1536**L**



C++11

- Celé číslo typu long long 1536**ll** alebo 1536**LL**

Čísla s pohyblivou rádovou bodkou

- v desatinnom tvare (-35.245)
- v semilogaritmickej tvare (1e12, -22.56E-11, 1E+3)
- pridaním f, F k celému číslu 15**f**, -321**F** 

Znakové konštanty



Znakové konštanty - znak uzavretý v apostrofoch

- ľubovoľný ('a', '2')
- L'a' alebo l'a' - konštantu typu `wchar_t`
- u'a' – konštantu typu `char16_t`
- U'a' – konštantu typu `char32_t`
- špeciálne znaky:

Názov znaku	ASCII symbol	C++ kód	Desiatkový/Hexa kód
Nový riadok	NL (LF)	<code>\n</code>	10/0xA
Horizontálny tabelátor	HT	<code>\t</code>	9/0x9
Vertikálny tabelátor	VT	<code>\v</code>	11/0xB
Krok späť	BS	<code>\b</code>	8/0x8
Návrat vozíka	CR	<code>\r</code>	13/0xD
Výstraha	BEL	<code>\a</code>	7/0x7
Opačné lomítko	<code>\</code>	<code>\\</code>	92/0x5C
Otáznik	<code>?</code>	<code>\?</code>	63/0x3F
Jednoduchá úvodzovka	<code>'</code>	<code>\'</code>	39/0x27
Dvojitá úvodzovka	<code>"</code>	<code>\"</code>	34/0x22

Reťazcové konštanty – znaky uzavreté v úvodzovkách

"Ahoj.\nSom na prednaske z C-cka\n"

Definícia konštanty

Konštanty typu const

const typ premenná

- konštanta, ktorú môžeme používať ako premennú, ale nemôžeme do nej priamo zapisovať
- ak v deklarácii chýba typ, predpokladá sa „int“

```
const float pi = 3.1415926535;
```

```
const max = 10000;
```

Symbolické konštanty (literálové)

- sú konštanty definované „#define“
- nie je to ozajstná konštanta
- **nahrádza identifikátor textom uvedeným za ním**
- zvykom je písať ich veľkými písmenami

```
#define PI 3.1415926535
```

```
#define OZNAM "Toto je oznam"
```

```
#define begin {
```

```
#define end ;}
```



Inicializácia premennej

- `int pocet = INT_MAX; // C, C++`
- `int pocet(INT_MAX); // C++`
- `int pocet = {INT_MAX}; // C++11`
- `int pocet {INT_MAX}; // C++11`
- `int pocet {};` // C++11, inicializácia na 0



Aritmetické operátory

- **+** pre sčítanie
- **-** pre odčítanie
- ***** pre násobenie
- **/** pre delenie (ak sú obidva operandy celočíselného typu, výsledkom je celočíselná časť podielu a zlomková časť sa zahodí)
- **%** pre výpočet modula, tj, zvyšku po delení (obidva operandy musia byť celočíselného typu)

```
int rozsah = 10;  
int i = 0;  
  
i = (i + 1) % rozsah;
```



Konverzie pri inicializácii a priradovaní



Typ konverzie	Potencionálny problém
Väčší typ pohyblivej rádovej bodky na menší typ pohyblivej rádovej bodky	Strata presnosti (platné číslice) <ul style="list-style-type: none">- hodnota môže byť mimo rozsah cieľového typu (výsledok nedefinovaný)
Typ pohyblivej rádovej bodky na celočíselný typ	<ul style="list-style-type: none">- Strata zlomkovej časti- Pôvodná hodnota môže byť mimo rozsah cieľovej hodnoty (výsledok nedefinovaný)
Väčší celočíselný typ do menšieho celočíselného typu	Pôvodná hodnota môže byť mimo rozsah cieľového typu (zvyčajne sa priradia iba spodné bity hodnoty)

Pretypovanie

```
int rozsah = 10;  
long celkom;
```

```
rozsah = (int)celkom; // C i C++  
rozsah = int(celkom); // len C++
```



```
rozsah = static_cast<int>(celkom); // pre smerníky, nerobí  
//kontrolu
```

```
dynamic_cast  
reinterpret_cast  
const_cast
```

auto deklarácia – len C++11

```
auto n = 100; // n je int  
auto x = 1.5; // x je double  
auto y = 1.3e12L; // y je long double
```

- Vhodné pre zložitejšie typy



Pole

Deklarácia (definícia) musí obsahovať:

- Typ hodnoty každého prvku
- Názov poľa
- Počet prvkov poľa

```
short tyzden[7];
```

- Indexovanie od 0 po PočetPrvkov-1



```
short matica[7][3];
```


Inicializácia poľa

```
int noha[4] = {3, 6, 8, 10};      // OK
int koleso[4];                    // OK
koleso[4] = {5, 6, 7, 9};        // Nesprávne, chyba typ
koleso = noha;                   // Nesprávne

int koleso[4] = {1,2};           // OK. Incializujú sa iba
                                // dve, zvyšok bude nula
int koleso[500] = {0};           // Všetky nulové

int koleso[] = {3, 6, 8, 10};     // OK
```



Inicializácia poľa C++2011

```
int noha[4] {3, 6, 8, 10}; // OK
```



```
int koleso[4] = {};           // Všetky nulové
```

```
int koleso[4]    {};          // Všetky nulové
```

```
long koleso[4] = {5, 6.3, 7, 9}; // Nesprávne
```

```
char znacka[4] = {'x', 'y', 'z', 568}; // Nesprávne
```

```
char znacka[4] = {'x', 'y', 'z', 121}; // OK
```

Reťazec

Postupnosť znakov, uložená v po sebe idúcich bytoch pamäti, ukončená nulou.

```
char meno[5] = {'k', 'a', 'r', 'o', 'l'}; // Pole, NIE reťazec
```



```
char budova[4] = {'d', 'o', 'm', '\\0'}; // Reťazec
```

```
char budova[4] = {"dom"}; // Reťazec
```

```
char budova[] = {"dom"}; // Reťazec
```

C++ - aj typ string

C++11 a ret'azec

```
char adresar[] {"c\\:temp\\test"};    // C++11  
char adresar[] {R"c\\:temp\\test"};    // C++11 - raw
```



```
wchar_t adresar[] = L"c\\:temp\\test";    // w_char ret'azec  
char16_t adresar[] = u"c\\:temp\\test";    // char_16 ret'azec  
char32_t adresar[] = U"c\\:temp\\test";    // char_32 ret'azec
```

Funkcie pre prácu s reťazcami

```
#include <string.h> // C, C++  
#include <cstring>  // C++11
```

- strcpy
- strlen
- strcat
- strrev
- strchr
- strstr

Štruktúra

```
struct student
{
    char meno[20];
    char priezvisko[60];
    int  rocnik;
};
```



```
student FRI;           // C++
struct student FRI;    // C
```

- V C iba dátové položky
- V C++ predstavuje druh triedy – môže mať i metódy

Inicializácia

```
student FRI = {"Novak", "Rene", 1 };
```

Lokálna štruktúra

```
void fun()  
{  
    struct student  
    {  
        char meno[20];  
        char priezvisko[60];  
        int  rocnik;  
    } FRI;  
  
    student PEDAS;  
}
```

`student FHV; // Nesprávne`



Bitové položky štruktúry

```
struct indikator  
{  
    unsigned int port : 6;  
    bool zapnuty : 1;  
    bool aktivny : 1;  
};
```

```
indikator siet = { 23, false, true };
```



union onion

```
union spolocny
{
    int ival;
    long lval;
    double dval;
};

spolocny.ival = 5;
```

0x2000



ival, lval, dval

Rozdiel medzi UNION a STRUCT je v tom, že v UNIONe sú všetky premenné uložené na rovnakom mieste v pamäti, ale v STRUCTe je pre každú premennú vyhradený osobitný pamäťový priestor

UNION je taký veľký, ako jeho najväčšia premenná.
STRUCT je taký veľký, ako súčet veľkostí jeho premenných.

<http://stackoverflow.com/questions/4003087/whats-the-major-difference-between-union-and-struct-in-c>

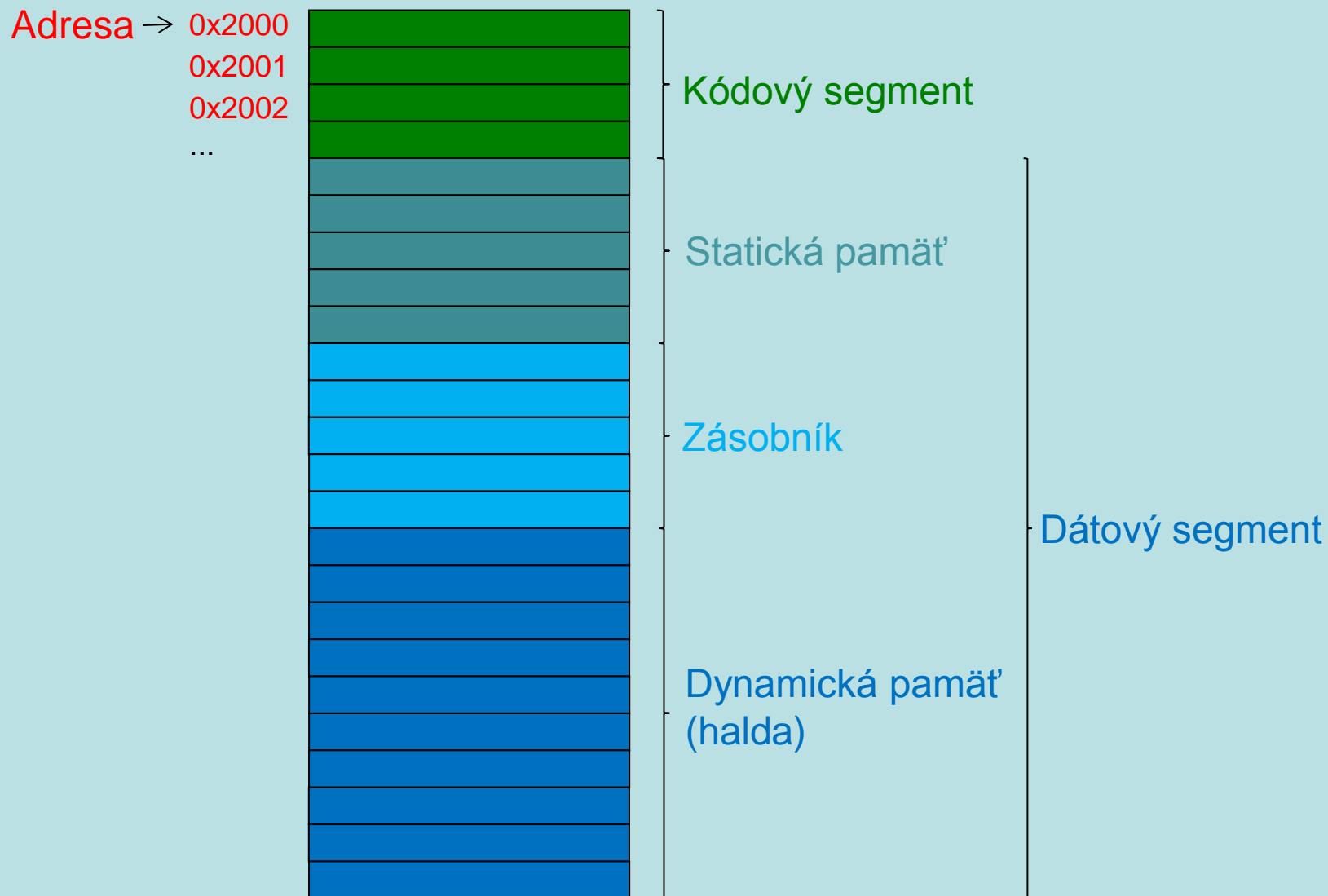
Anonymné union

```
union
{
    int ival;
    long lval;
    double dval;
};

ival = 5;
```



Pamät'



Smerníky

- smerník je premenná, ktorá obsahuje adresu inej premennej
- pomocou smerníka môžeme ku premenným pristupovať tzv. nepriamo
- adresu premennej **x** získame výrazom **&x**
- ak premenná **px** je deklarovaná ako smerník na int potom obsah premennej na ktorú ukazuje **px** získame výrazom ***px**
- výraz ***px** môžeme používať na pravej aj na ľavej strane priradovacieho výrazu



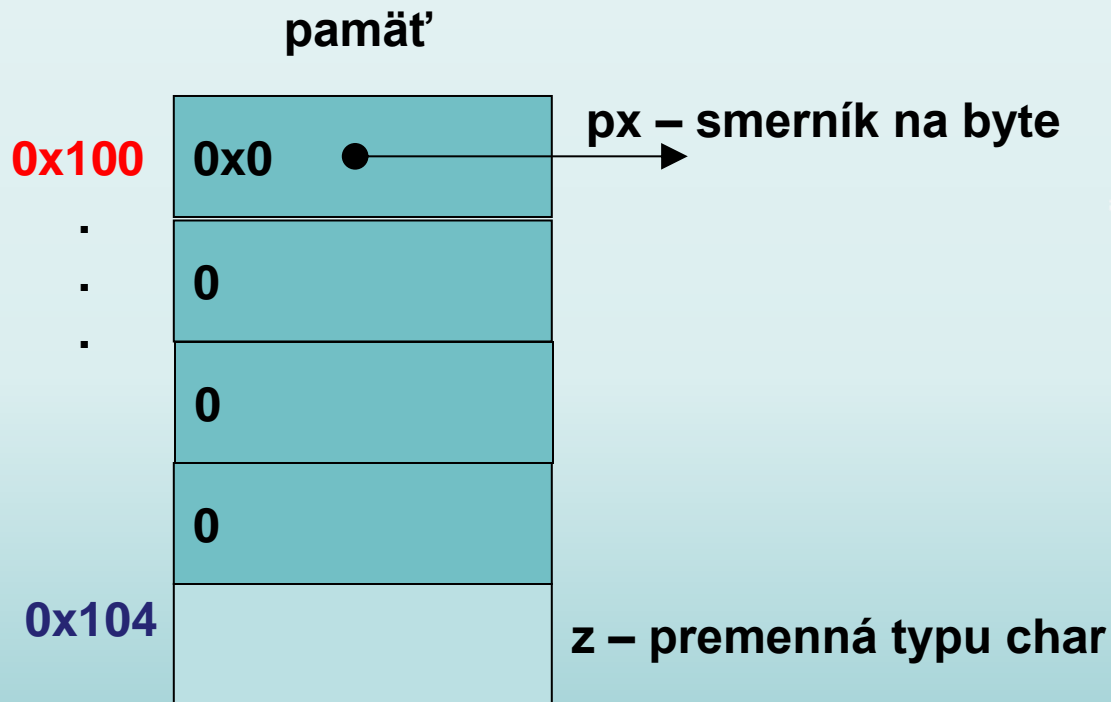
Smerníky - definícia

char *px;

sizeof(px) ... 4

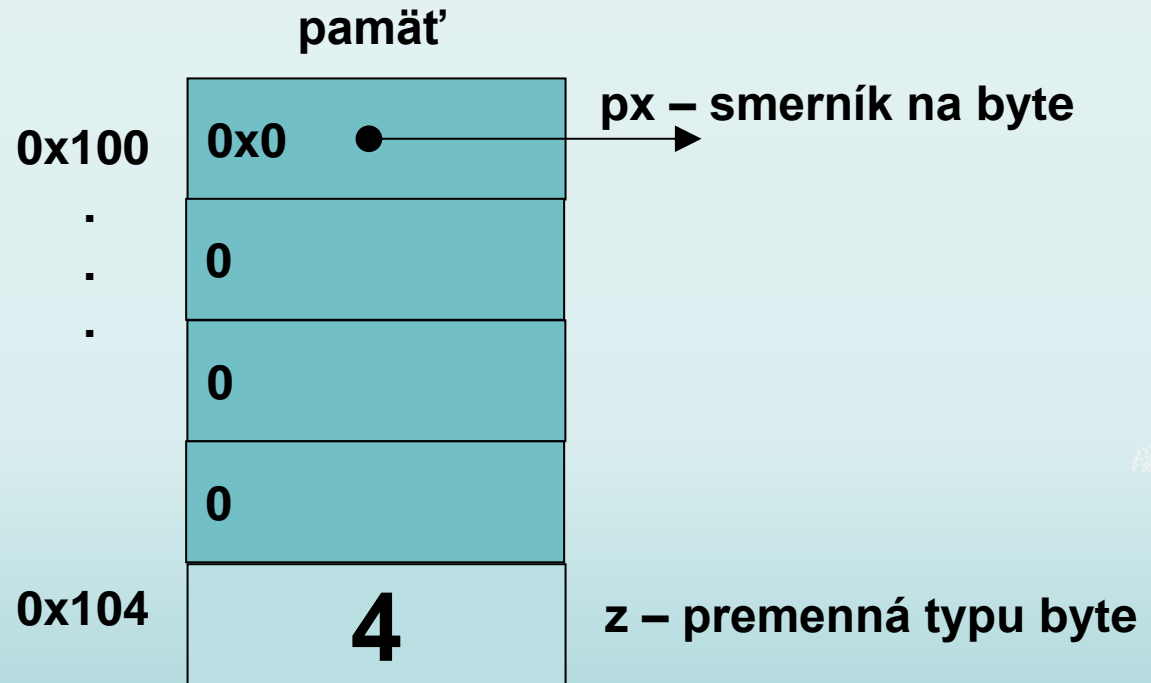
char z;

sizeof(z) ... 1



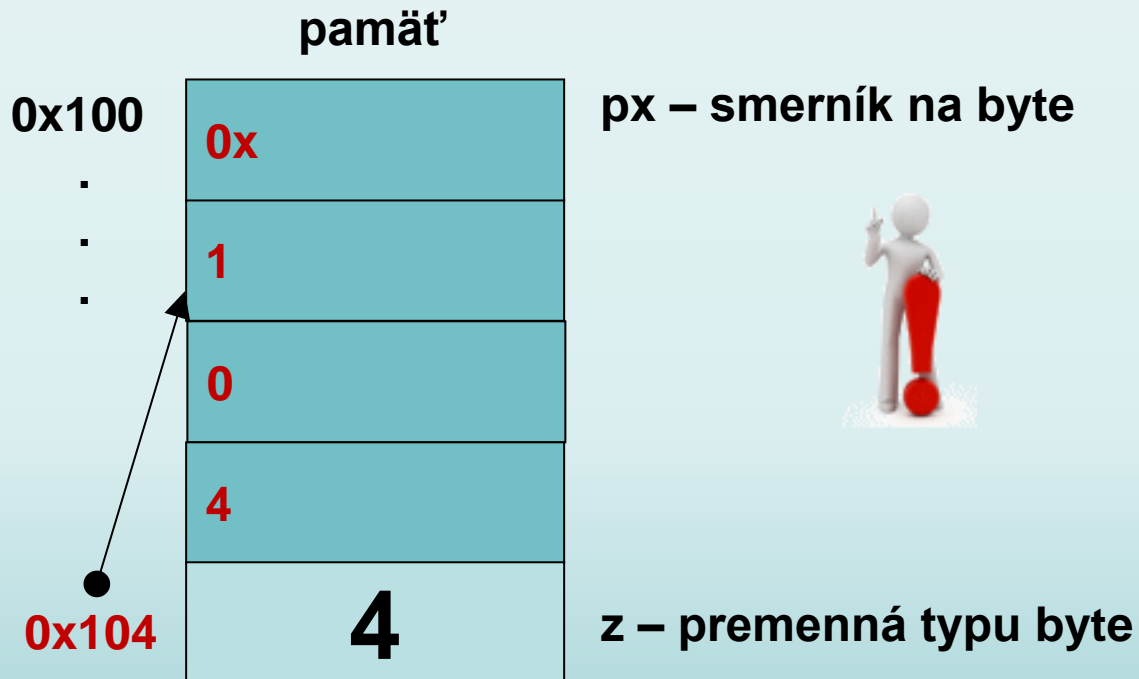
Smerníky – práca so smerníkom

z = 4;



Smerníky – priradenie adresy

px = &z;



Smerníky – priradenie hodnoty

