

# NSWI170 – Počítačové systémy

Tomáš Faltín

# Ukazatele (pointers) v C (1/2)

- Jde o typ proměnných (podobně jako int, double, pole, ...)
- `typ *jmeno`
  - Překladač kontroluje/zná typ, na který se ukazatel odkazuje
- Příklady:
  - `int *p1 = ...;` // ukazatel **p1** na typ **int**
  - `char *p2 = ...;` // ukazatel **p2** na typ **char**
  - `int **p3 = ...;` // ukazatel **p3** na typ **ukazatel na int**
- Příklady použití:
  - při dynamické alokaci (ne v Arduino)
    - malloc/new vrátí ukazatele
  - práce s poli/řetězci
    - pole je vlastně ukazatel na začátek
  - výstupní parameter z funkce
    - v C se všechny parametry předávají hodnotou

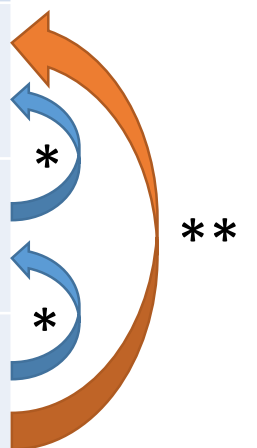
# Ukazatele (pointers) v C (2/2)

- Speciální operátor **&** vrací adresu proměnné
  - `int x = 3;`  
`int *px = &x;`
    - p\_x nyní obsahuje adresu proměnné x
    - jinak řečeno: p\_x „ukazuje“ na proměnnou x
  - **Pozor** v C++ má & i jinou funkci – reference
- Speciální hodnota `nullptr`, když „neukazují na nic“
  - `int *px = nullptr;`
- Operátor **\***
  - Přístup na hodnotu uloženou na adrese, která je v pointeru
  - `int x = 3;`  
`int *px =`  
`Serial.print(*px);`

# Ukazatele (pointers) v paměti

```
int main() {  
    int i = 2;  
    int *pi = &i;  
    int **ppi = &pi;  
    Serial.print(i); // 2  
    Serial.print(pi); // 102  
    Serial.print(*pi); // 2  
    Serial.print(ppi); // 104  
    Serial.print(*ppi); // 102  
    Serial.print(**ppi); // 2  
}
```

Adresa	Obsah	(Proměnná)	Operátor *
...			
100	2	i	*
101			
102	100	pi	*
103			
104	102	ppi	*
105			
...			



**Q:** Co se stane pokud bychom zavolali: `Serial.print(*i)`

# Q: Serial.print(\*i);

```
int i = 2;  
Serial.print(*i);
```

- Program vrátí hodnotu, která leží na adrese 2, kde může ležet cokoliv ☠
- Poznámka: překladač vás to nenechá udělat, jelikož ví, že se jedná o číslo, na kterém nesmíte volat operátor \*

Adresa	Obsah	(Proměnná)	Operátor *
0			
1			
2	???		
....			
100	2	i	*
101			
102	100	pi	
103			
104	102	ppi	
105			
....			

# Ukazatele (pointers) v reálném světě



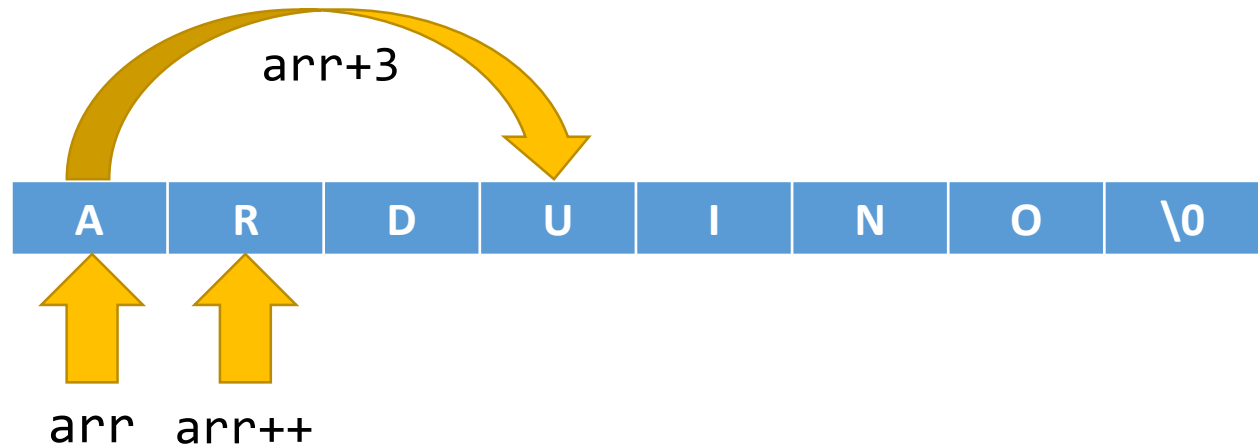
# Řetězce v C

- Pole jsou (z pohledu překladače) jen zvláštní druh ukazatele, který ukazuje na první element
  - `char [] ~ char *`
  - `char arr[] = { 1, 2, 3 }; assert(arr[0] == *arr);`
- řetězec = pole znaků `char[]` zakončené znakem `'\0'`
  - Př.: `const char str*="ARDUINO"`
- Práce s řetězcí pomocí ukazatelů

A	R	D	U	I	N	O	'\0'
---	---	---	---	---	---	---	------

# Práce s ukazateli

- Ukazatelová aritmetika (+, -, ++, --, ...)
  - `*arr = arr[0]`
  - `*(arr + 3) = arr[3]`
  - `++arr, arr--, *arr++`





# Počítání délky řetězce

```
size_t len1(const char *str) {  
    size_t i = 0;  
    while(str[i] != '\0') { ++i; }  
    return i;  
}
```

```
size_t len2(const char *str) {  
    size_t i = 0;  
    while(*str++) { ++i; }  
    return i;  
}
```

```
int main() {  
    print(len1("Arduino"), len2("Arduino")); // 7, 7  
}
```

# Úkol 1: funkce na porovnávání řetězců

- Funkce dostane dva parametry – řetězce a vrátí pořadí (**číslované od 1**) prvního znaku, který se liší, pokud jsou řetězce stejné, vrátí 0
- Pomocí ukazatelové aritmetiky
- Musí fungovat i v případě nullptr, prázdných řetězců

```
str_cmp("jedna", "jed") == 4
```

```
str_cmp("jedna", "dva") == 1
```

```
str_cmp(nullptr, "dva") == 1
```

```
str_cmp(nullptr, "") == 1
```

```
str_cmp(nullptr, nullptr) == 0 // to samé i pro ""
```

```
str_cmp("tri", "tri") == 0
```

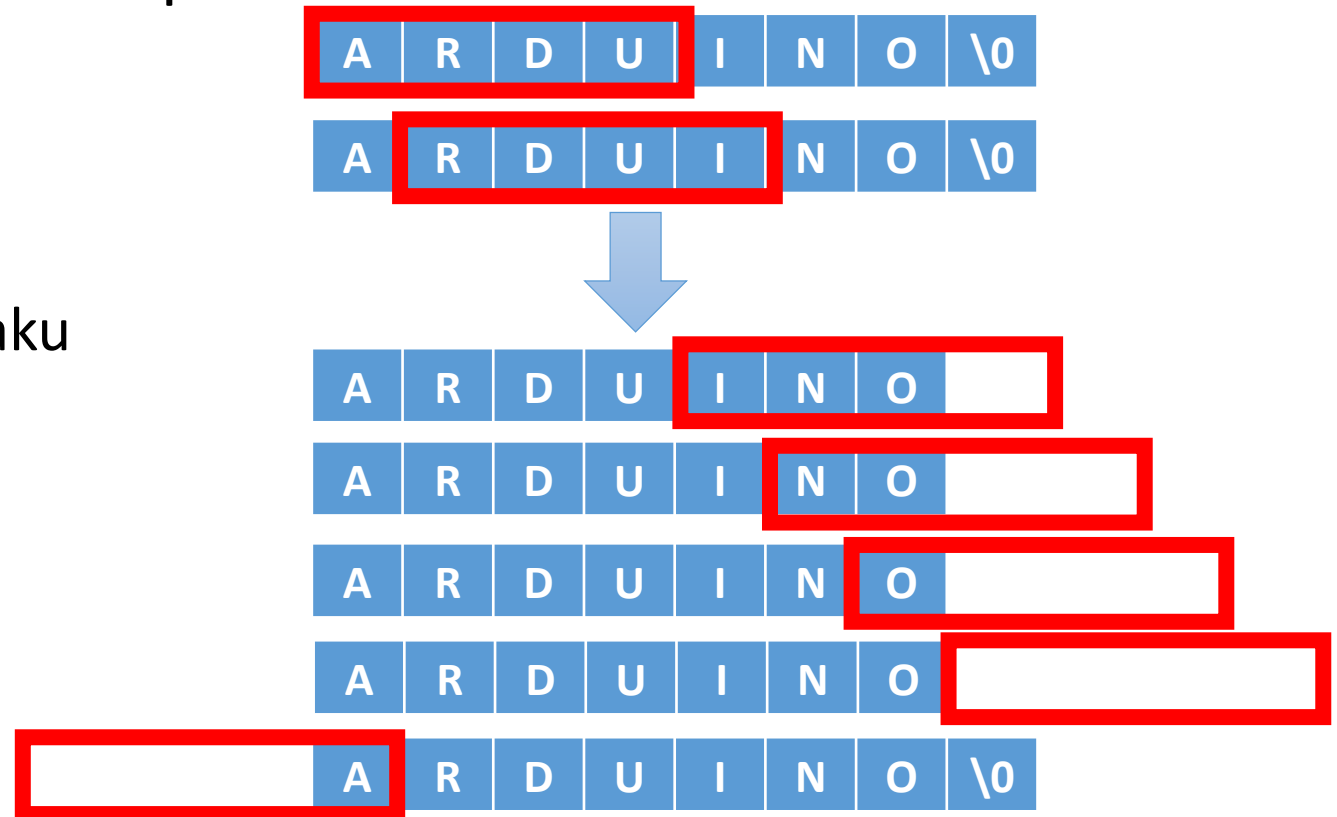
## Úkol 2: Zobraz řetězec na displeji

- Funkce, která umí zobrazit zadaný řetězec na displeji
- Pokud se řetězec nevejde, zobrazí jen první 4 znaky



# Úkol 3: Zobraz celý řetězec na displeji

- Funkce, která cyklicky zobrazuje **celý** řetězec na displeji
- Pokud se řetězec nevejde, postupně se řetězec posouvá
- Řetězec se nechá úplně zmizet a pak se začne zobrazovat znova
  - Řetězec utíká doleva
- Funkce bere 2 parametry:
  - Řetězec
  - Délka zobrazení jednoho znaku



# Domácí úkoly

- Nahrád do SISu zdroják obsahující funkce pro úkoly 1 a 3
- Do 14 dnů
- Podmínky:
  - Funkční
  - **Zohlednit zpětnou vazbu z posledně!**
  - Rozdělené do funkcí
  - Srozumitelně pojmenované konstanty/funkce
  - **Objektový návrh** funkcí = funkce, které k sobě patří mají stejný prefix