

Arduino 7-segmentový displej Běžící zpráva

NSWI170: Lab 06

Patrik Dokoupil

Credit: Martin Kruliš

Shrnutí 5. úlohy

- Problémy:
 - Ukládání stavů do znaků
 - State == 's' apod. lze nahradit za enum
 - `enum class State { stopped, running };`
 - Nastavení tečky pomocí +/-128
 - Flagy typicky nastavujete pomocí bit. operací
 - Problémy s časováním (špatné výpočty, zvyšování po 0.1)
 - Mnohokrát opakované volání **digitalRead**
 - Ukládání časů do `int` (na fyzickém Arduino overflow po cca 30s)

Pointer (ukazatel)

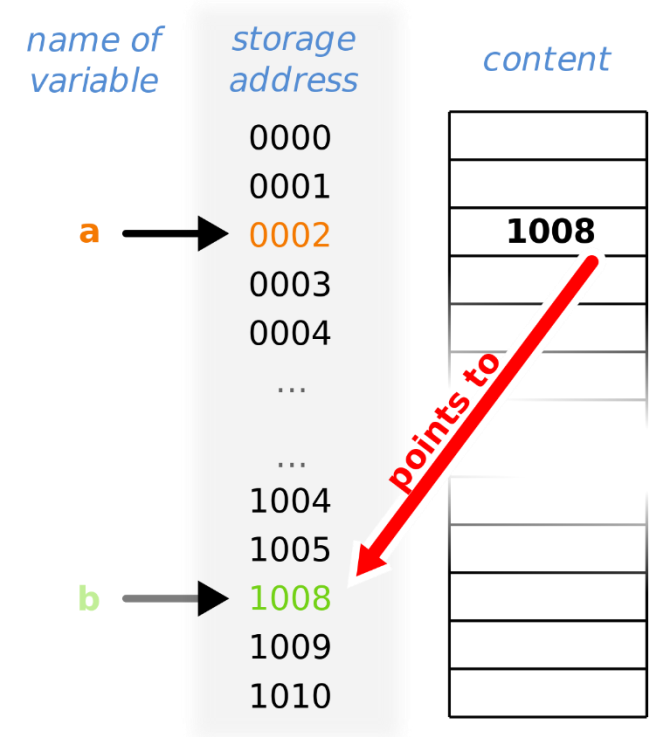
- Proměnná která obsahuje adresu někam do paměti (typicky na další proměnnou)
- Syntaxe: **type * p = <pointer expression>; // p je ukazatel na typ type**
 - Př. **int * p; // ukazatel na int**
- <pointer expression> je výraz pro inicializaci ukazatele, typicky jeden z následujících:
 - **nullptr** – neukazuje nikam (C++ „náhrada“ za Cčkové ~~NULL makro~~)
 - Adresa existující proměnné (př. **&a**)
 - Výpočet z jiného pointeru (pointerová aritmetika)
 - Dynamická alokace (**malloc/new**) – bez ní se však obejdeme

Pointer (ukazatel) - ukázka

```
int b = 0; // alokuje místo pro b (zde adresa 1008)
           // uloží tam 0

int * a = &b; // a je ukazatel na b
              // alokuje místo pro a (zde adresa 0002)
              // uloží tam adresu proměnné b (1008)
              // * slouží k deklaraci pointeru
              // & bere adresu tj. není to reference, ale
              // „address-of“ operátor

int x = *a + 1; // alokuje místo pro x, uloží tam
               // hodnotu v a zvětšenou o 1
               // * je v tomto kontextu dereference
```



Zdroj: wikipedia.org

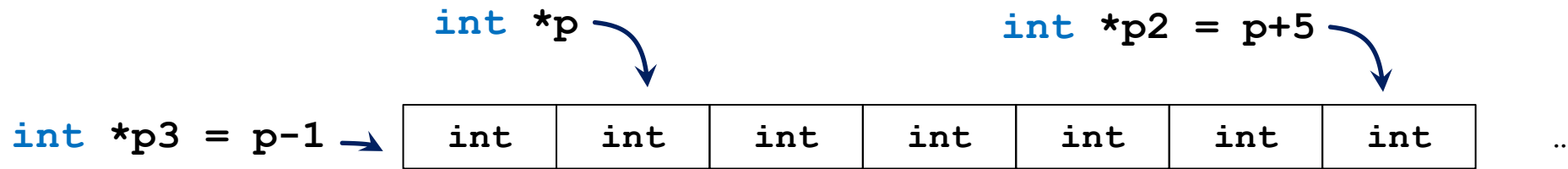
Pointerová aritmetika

- Adresu na kterou pointer ukazuje lze posouvat

- `int * p2 = p + 5;`
- `int * p3 = p - 1;`

posun o 5 kroků,
velikost kroku je
určena typem (posunete
se o `5 * sizeof(int)`
bytů)

Posun o -1 krok, lze také `--p`



- Vztah k polím: proměnná pro pole se chová +- jako pointer na první prvek v poli, tedy

- `int buf[10];` // 10 za sobě jdoucích `intů` v paměti
- `buf[i]` odpovídá `*(buf + i)`

Používejte `*` pro
dereferencování pointerů, `[]`
pro indexování polí


String

- Stringy v C

- Pole znaků ukončené nulou (poslední znak je \0)

- `const char * str = " Arduino Uno";`

`*str`



A	r	d	u	i	n	o		U	n	o	\0
---	---	---	---	---	---	---	--	---	---	---	----

Pro stringové literály dostanete \0 na konci automaticky

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

Co dělá tato funkce?

Časté chyby

- Neinicializovaný pointer může ukazovat na náhodné místo v paměti

- `int * p;`

Pointery raději hned inicializujte: `int * p = nullptr;`

- Pointer mimo hranice pole

- `int buf[10];`

- `int * p = buf + 10; // 1 prvek za buf`

- `*p = 100; // zápis do paměti za pole, kde může být +- cokoliv, klidně jiná proměnná`

- Stringy v polích fixní délky

- `char buf[16];`

- Je to možné, ale musíte si dát pozor na overflow (nezapomínejte na místo pro `'\0'`)

Cvičení

- T1 – Zobrazení textové zprávy
 - Zobrazte na displeji textovou zprávu
 - Max 4 znaky, zbytek textu ignorujte
 - Upravte si class pro displej aby umožňovala nastavit zprávu
 - `void set (const char * msg) ;`
 - Stejně jako minule, použijte multiplexing
 - Masky pro znaky jsou k dispozici v ReCodExu u poslední úlohy

Cvičení

- T2 – Zobrazte běžící zprávu
 - Zpráva může být delší než 4 znaky, pak ji na displeji „rotujte“ viz ukázka
 - Rychlost rotace (po kolika MS se posunete) bude parametr

			A	Začátek
		A	r	
	A	r	d	
A	r	d	u	
r	d	u	i	
d	u	i	n	
u	i	n	o	
i	n	o		
n	o			
o				Konec

Domácí úloha*

- Běžící zpráva
 - Úkolem je zobrazovat běžící zprávu, podobně jako v T2
 - Zpráva jako taková, bude zadána Sériovým portem
 - Zobrazíte vždy celou zprávu, jakmile skončí, zjistíte, jestli máte novou zprávu - pokud ano, zobrazíte ji, jinak zobrazíte opět starou
 - Očekává se použití Cčkových stringů, nikoliv `std::string`
 - Detaily zadání (+ starter pack) v **ReCodExu**

Finální úloha**

- Simulace hrací kostky pro Dungeons&Dragons, Detailní zadání na [webu](#)
- Finální deadline je +- konec června (přesný termín bude vypsán)
 - Optimální deadline je +- konec května
- Odevzdání do ReCodExu + **osobní demonstrace**
 - Bude několik (3) termínů, kdy je možné Vaše řešení předvést
 - Poslední cvičení (18.5.)
 - Dále budou 2 termíny během června, jeden někdy na začátku, druhý v posledním týdnu (s tím souvisí, že **finální deadline je dán tímto posledním termínem**)
 - Alespoň 24h před termínem je nutné odevzdat řešení do ReCodExu
 - Na demonstraci si vezměte vlastní Arduino
 - **Dodržujte** všech [5 pravidel](#) která zazněla během cvičení
 - Použít můžete libovolný kus kódu který jste napsali v úlohách během semestru
- Všech 6 standardních úloh musíte mít ve stavu „Accepted“