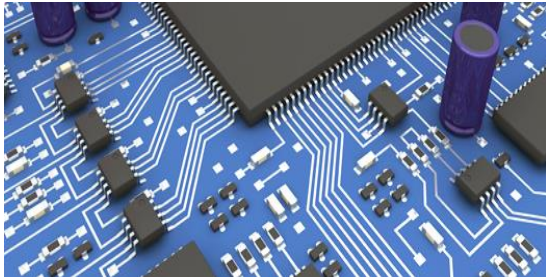


Aplikace Embedded systémů v Mechatronice



Michal Bastl
A2/713a

Aplikace Embedded systémů v Mechatronice

Obsah přednášky:

- Opakování
- Pointery v C
- pole a řetězce
- předání funkci referencí
- Vlastní datové typy typedef
- Struktury
- Ukázka
- Hardware poznámky



Opakování

Postup při vytvoření vlastní funkce?


Postup při vytvoření knihoven?

Co patří do hlavičkového souboru?

Pointery/ukazatele

- Ukazatele v jazyce C slouží k přístupu do paměti a manipulaci s adresou.
- Celá věc v C funguje tak, že existují speciální proměnné, které uchovávají adresu v paměti.
- V C můžete pointer vytvořit příkazem `typ* proměnná`
- právě znak `*` určuje, že se bude jednat o ukazatel na příslušný datový typ
- pokud chci získat adresu proměnné používám referenční operátor `&`
- dereferenční operátor `*` slouží k získání hodnoty uložené na adrese

ADD	VAL
0x00	10
0xff	0x00



Pointery/ukazatele

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main()
{   int c;
    int* p_c;

    c = 10;
    p_c = &c;

    printf("Na adrese 0x%p je hodnota%d\n",p_c,*p_c);
    return 0;
}
```

operátor reference &c vrací adresu paměti
operátor dereference *p_c vrací hodnotu
uloženou na adrese
symbol *p_c slouží současně pro deklaraci
pointeru

>>Na adrese 0x0060FF08 je hodnota 10

Pointery-předání funkci

```
#include <stdio.h>
#include <stdlib.h>
```

```
void prohod(int* a, int* b);
```

```
int main(){
    int jedna;
    int dva;

    jedna = 1;
    dva = 2;
    prohod(&jedna, &dva);
    printf("jedna = %d; dva = %d\n", jedna, dva);
    return 0;
}
```

```
void prohod(int* a, int* b){
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

```
>>jedna = 2; dva = 1
```

operátor reference &c vrací adresu paměti
operátor dereference *p_c vrací hodnotu
uloženou na adrese
symbol *p_c slouží současně pro deklaraci
pointeru

```
void prohod(int a, int b){
    int tmp = a;
    a = b;
    b = tmp;
}
```



NEFUNGUJE!!

Pointer vs. pole

```
#include <stdio.h>
#include <stdlib.h>
```

```
void uloz_do_pole(int pole[], int index, int cislo);
```

```
int main() {
```

```
    int ciska[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
    printf("%d\n", ciska[7]);
```

```
    uloz_do_pole(ciska, 7, 3);
```

```
    printf("%d\n", ciska[7]);
```

```
    return 0;
```

```
}
```

```
void uloz_do_pole(int pole[], int index, int cislo){
    pole[index] = cislo;
}
```

```
>>7
```

```
>>3
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main() {
```

```
    char abc[] = "Pointery jsou fajn!";
```

```
    char* p_abc = abc;
```

```
    while(*p_abc != '\0'){
        printf("%c", *p_abc);
        p_abc++;
    }
```

```
    return 0;
```

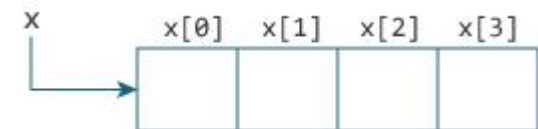
```
}
```

Pole a pointery spolu v C souvisí. Pokud předám funkci pole, provádím to vždy referencí. Proto změny, které ve funkci provedu, v poli zůstanou zachovány. Toto předání referencí proběhne a nemusíme se o to snažit.

Pole v C je ukazatel na místo v paměti, kde pole začíná.

Proto:

ciska[1] a *ciska + 1 vrací stejný výsledek



Aritmetika pointerů

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

int main() {

    int16_t pole[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int16_t *p_prvni, *p_posledni;
    p_prvni = pole;
    p_posledni = pole + 9;

    if(p_posledni > p_prvni){

        printf("Adresa %d \n", p_prvni);
        printf("Adresa %d \n", p_posledni);
        printf("Prvni %d \n", *p_prvni);
        printf("Posledni %d \n", *p_posledni);
        printf("Vysledek %d \n", p_posledni - p_prvni);

    }

    return 0;
}
```

S pointery jde počítat. Lze k nim přičítat celá čísla. Lze je mezi sebou porovnávat a také přičítat a odčítat mezi sebou. Smysluplné výsledky dostaneme například pokud máme dva ukazatele v jednom bloku paměti. Je třeba mít na paměti, že dochází ke srovnávání adres a tedy porovnání v příkladu `p_posledni > p_prvni` říká, že `p_posledni` je „dále“ v bloku paměti. Rozdíl v příkladu je devět bloků příslušného datového typu. Tedy dle adres 18 bajtů. Kód `p_prvni++` tedy posune ukazatel o dva bajty. Hodnotu do které se ukládá `int16`.

```
>>Adresa 6356724
>>Adresa 6356742
>>Prvni 1
>>Posledni 10
>>Vysledek 9
```


typedef – uživatelské datové typy

V jazyce C je možné vytvořit uživatelský datový typ používá se klíčového slova typedef

Příklad samozřejmě nemá valný smysl, tato možnost se s výhodou používá např. právě při tvorbě struktur v C

```
#include <stdio.h>
#include <stdlib.h>

int main() {

    typedef int mujInt;

    mujInt a, b;

    a = 10;
    b = 20;

    mujInt c = a + b;

    printf("%d", c);

    return 0;
}
```

Struktury

Struktura je zjednodušeně datový typ, do které uzavřeme další datové typy, které s tímto typem nějak abstraktně souvisí. Například každý uživatel má jméno, věk atd.

Struktura může uchovávat různé datové typy a pole.

Strukturu lze vytvořit různým zápisem, ale vřele doporučujeme držet se tohoto zápisu a vytvořit strukturu jako nový datový typ. V příkladu je umístěna do globálního prostoru.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
typedef struct{
    char  jmeno[25];
    int   vek;
    int   vyska;
} clovek;
```

```
int main() {
```

```
    clovek Petr = {"Petr Novak", 25, 178};
    clovek Michal;
```

```
    Michal.vek = 16;
    Michal.vyska = 193;
```

```
    strcpy(Michal.jmeno, "Michal Novak");
```

```
    printf("Petr ma %d let\n", Petr.vek);
    printf("Michal se jmenuje %s", Michal.jmeno);
```

```
    return 0;
```

```
}
```

```
>>Petr ma 25 let
```

```
>>Michal se jmenuje Michal Novak
```

Struktury

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "types.h"
```

```
int main() {

    if (LIMIT > 9){

        clovek Petr = {"Petr Novak", 25, 178};
        clovek Michal;

        Michal.vek = 16;
        Michal.vyska = 193;

        strcpy(Michal.jmeno, "Michal Novak");

        printf("Petr ma %d let\n", Petr.vek);
        printf("Michal se jmenuje %s", Michal.jmeno);
    }
    return 0;
}
```

```
#ifndef TYPES_H_INCLUDED
#define TYPES_H_INCLUDED
```

```
#define LIMIT 10
```

```
typedef struct{
    char    jmeno[25];
    int     vek;
    int     vyska;
} clovek;
```

```
#endif // TYPES_H_INCLUDED
```

Zadání domácí úlohy

- Vytvořte textovou kalkulačku
- Textový vstup z terminálu v MCU zpracujete a vrátíte požadovaný výsledek
- uvažujte operátory +; -; *; /;
- pracujte s celými čísly (int) 0..99

Ukázka:

```
>>10+25  
>>výsledek: 35
```

```
>>10/5  
>>výsledek: 2
```



Hardware

