Informační a komunikační technologie

Adresy a ukazatele v C

David Weber
Kabinet K13

weber3@spsejecna.cz

Připomenutí

- Jaký je rozdíl mezi *adresou* a *hodnotou* proměnné?
  - **Hodnota** – hodnota uložená v paměťové buňce proměnné
  - **Adresa** – číslo paměťové buňky proměnné
- Pro výpis jsme používali znak `&`.

- Předpona `0x` značí číslo zapsané **hexadecimálně** (v C, ale i jiných jazycích).

Příklad na úvod I

Mějme program obsahující funkci pro prohození hodnot proměnných u a v.

```
void swap(int u, int v) {
    int temp = u;
    u = v;
    v = temp;
}
int main(void) {
    int a = 5;
    int b = 10;
    swap(a, b);
    printf("Hodnoty a, b: %d, %d", a, b);
    return 0;
}
```

Příklad na úvod II

Jaký bude výstup předešlého programu?

(a) `Hodnoty a, b: 5, 10`

(b) `Hodnoty a, b: 10, 5`

Příklad na úvod II

Jaký bude výstup předešlého programu?

(a) `Hodnoty a, b: 5, 10`

(b) `Hodnoty a, b: 10, 5`

Příklad na úvod II

Jaký bude výstup předešlého programu?

(a) `Hodnoty a, b: 5, 10`

(b) `Hodnoty a, b: 10, 5`

⇒ nijak jsme si nepomohli 😰

V čem je problém?

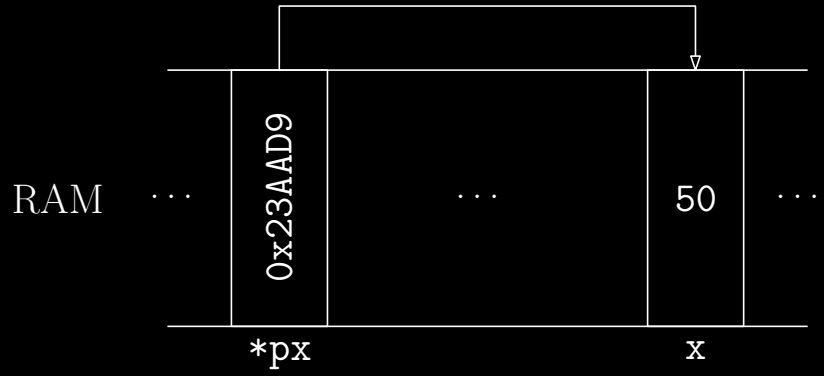
- Parametry funkci předáváme tzv. **hodnotou**.
  - Hodnoty proměnných `a` a `b` jsou zkopírovány a při volání funkce `swap` jsou nově *na zásobníku* deklarovány proměnné `u` a `v`.
  - $\Rightarrow$ funkce v konečném důsledku prohodí hodnoty proměnných `u` a `v`, nikoliv `a` a `b`
- $\Rightarrow$ mohli bychom vyřešit předáním "odkazů" na původní proměnné.

Ukazatel (pointer) I

- **Datový typ** uchovávající adresu v paměti **určitého datového typu** (existují i generické ukazatele, ale ty nebudeme řešit 😳).
- Při deklaraci je třeba uvést datový typ hodnoty (tím kompilátoru říkáme, jak se má interpretovat místo v paměti, kam ukazuje), jejíž adresu ukazatel uchovává, a znak * (pro odlišení od deklarace standardní proměnné).
- Jedná se vždy o **kladné celé číslo** (formátová specifikace %p).

```
int main(void) {
    int x = 50;
    int *px = &x;
    printf("Hodnota x: %d\nAdresa x: %p", x, px);
    return 0;
}
```

Ukazatel (pointer) II



Dereference

- Ekvivalentně lze deklaraci ukazatele zápisem `int* p` (mezera mezi `*` a `p`).
- Co když máme adresu, ale ne samotnou proměnnou? ⇒ využijeme tzv. **operátor dereference** `*`.
  - Umožňuje nám odkázat se přímo na **hodnotu** v paměťové buňce ležící na **adrese ukazatele**.

```
int main(void) {
    int i = 50;
    int *p = &i;
    *p = 100;           // same as i = 100;
    printf("%d", i);    // prints out "100"
    return 0;
}
```

Původní problém

Jak bychom měli upravit původní program, aby fungoval?

```
void swap(int u, int v) {
    int temp = u;
    u = v;
    v = temp;
}
int main(void) {
    int a = 5;
    int b = 10;
    swap(a, b);
    printf("Hodnoty a, b: %d, %d", a, b);
    return 0;
}
```

Řešení

- Je potřeba předat funkci `swap` adresu proměnných `a` a `b` (tzv. předání **referencí**).
- Uvnitř těla funkce pak provedeme dereferenci ukazatelů.

```
void swap(int *u, int *v) {
    int temp = *u;
    *u = *v;
    *v = temp;
}
```

Pole a ukazatele I

- Již jsme si zmínili, že pole je struktura uchovávající více prvků **stejného datového typu**.
- Na jednotlivé buňky pole jsme se při načítání hodnot z konzole odkazovali pomocí jejich adresy v paměti.

```
int arr[10];
for (int i = 0; i < 10; i++) {
    if (scanf("%d", &arr[i]) < 1) {
        printf("ERROR.");
        return -1;
    }
}
```

Pole a ukazatele II

- Samotný „výraz“ `arr` jsme používali (zatím) vždy pouze s indexem při odkazování na konkrétní prvek (např. `arr[i]`).
- Co ovšem představuje sám o sobě? ⇒ ukazatel na počátek pole! (Tzn. prvek na nultém indexu.)

```c
int main() {
    int arr[] = { 1, 2, 3 };

    // arr and &arr[0] are identical
    printf("%p %p", arr, &arr[0]);
    return 0;
}
```

Příklad (hledání prvku v poli) I

```
unsigned int getElementIndex(int arr[], int size, int
   element) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == element)
            return i;
    }
    return -1;
}
```

Příklad (hledání prvku v poli) II

- Funkce `getElementIndex` přijímá jako parametr pole `arr`, které je předáváno **referencí**.
- Předávání hodnotou by mohl být problém, neboť pro velké struktury (řádově např. statisíce prvků) bychom museli všechny prvky kopírovat.
- Lze též psát `int* arr`.

Pole a ukazatele III

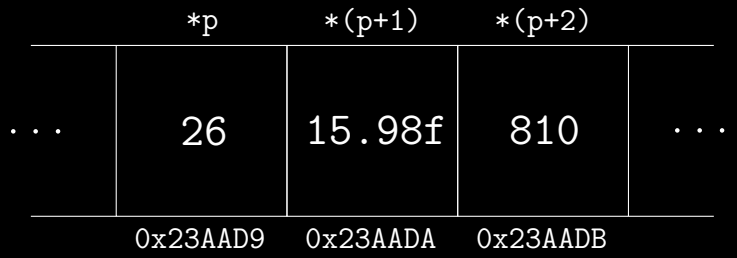
```c
int main(void) {
    int arr[5];
    int *p = arr;
    printf("%p, %p", sizeof(arr), sizeof(p));
    return 0;
}
```

- Je dobré si uvědomit, že byť `arr` ukazuje na počátek pole, přesto je zde rozdíl oproti ukazateli `p`.
- $\Rightarrow$ `sizeof(arr)` vypíše velikost pole `arr` v bytech, zatímco `sizeof(p)` vypíše velikost ukazatele `p`!

Aritmetika s ukazateli I

- Ukazatele jsou číselné proměnné $\Rightarrow$ lze s nimi manipulovat pomocí $+$, $-$.
- Nelze např. sčítat/odčítat dvojici ukazatelů, tj. `p1 + p2`
- Lze psát např. `p++`, `p--`, `p -= 4`, ...
- **Inkrementace**, resp. **dekrementace** probíhá vždy o násobky velikosti datového typu, na který ukazatel ukazuje (nikoliv přímo o danou hodnotu)!

Aritmetika s ukazateli II



Příklad

Výpis prvků pole.

```
void printArray(int arr[], unsigned int size) {
    int *p = arr;       // First element address
    for (int i = 0; i < size; i++) {
        printf("%d", *(p+i));
    }
}
```

Otázky?