Strukturované typy a ukazatele

Úvod do programování 1

Tomáš Kühr

Motivace

- Se základními datovými typy si sice vystačíme
- Někdy to ale může být nepříjemně nepřehledné
- Příklady:

```
long double soucet(const long double data[], int delka);

int test_splnen;
if (test(data)>10) test_splnen = 1; else test_splnen = 0;

int dnes[3]; // datum
dnes[0] = 5; dnes[1] = 11; dnes[2] = 2014;

char jmeno_osoba[] = "Petr";
int narozen_osoba[3] = {30, 4, 1995};
```

Vlastní datové typy

- Definujeme pomocí konstrukce `typedef`
- Definice typu obecně:
  `typedef` *`zápis_ve_tvaru_definice_proměnné`*`;`
- Příklady definice typu:

```
typedef int cele_cislo;
typedef int pole_deseti_cisel[10];
typedef unsigned long ULong;
typedef const ULong CULong;
typedef char string[];
```

- Příklady použití typu (definice proměnné):

```
cele_cislo pocet;
pole_deseti_cisel moje_cisla;
ULong velke_cislo;
string jmeno = "Ales";
```

Výčtové datové typy

- Definujeme obvykle pomocí `typedef` a `enum`
- Používá se pro definici spolu souvisejících celočíselných konstant
- Definice typu obecně:

```
typedef enum{
    jmeno1 = hodnota1,
    ...
    jmenoN = hodnotaN
} jmeno_typu;
```

- Příklad definice:

```
typedef enum{
    TRUE = 1, FALSE = 0
} boolean;
```

- Příklad použití typu:

```
boolean vysledek = TRUE;
```

Příklady – výčtové typy

```
typedef enum{
    FALSE, TRUE
} boolean;

typedef enum{
    Po, Ut, St, Ct, Pa, So = 10, Ne
} den;

boolean splneno;
boolean splneno = 1;
boolean splneno = TRUE;
den ZP1 = St;

splneno = ZP1 * 5 - TRUE + Ut * So;
```

Strukturované datové typy

- Definujeme obvykle pomocí `typedef` a `struct`
- Uložení více souvisejících hodnot do 1 proměnné
- Hodnoty mohou být různých typů (rozdíl od pole)
- Definice strukturovaného typu obecně:

```
typedef struct {
    typ1 jmeno_clenu1;
    ...
    typN jmeno_clenu1;
} jmeno_typu;
```

- Příklad definice:

```
typedef struct {
    char den;
    char mesic;
    short rok;
} datum;
```

Příklady – strukturované typy

- Strukturované proměnné definujeme obvyklým způsobem:

```
datum narozen;
datum narozen = { 24, 3, 1972 };
```

- Přístup ke členům proměnné pomocí operátoru tečka:

```
datum muj_den, narozen, dnes;
int vek;
…
muj_den.den = 24;
muj_den.mesic = 1 + (muj_den.mesic + 5) % 12;
vek = dnes.rok - narozen.rok;
```

Struktury vs. pole

- **Struktury** mohou obsahovat položky různých datových typů
- Používáme je pro uložení různých informací o nějakém celku
- Struktury lze kopírovat přiřazením:
```
narozen = dnes;
```
- **Pole** mohou obsahovat pouze položky jediného typu
- Používáme je pro uložení více datových položek, se kterými chceme pracovat jednotným způsobem
- Pole **nelze** kopírovat přiřazením:
```
int cisla[] = { 2, 5, 11, 17, 23 };
int data[5];

data = cisla;
```

Složitější struktury

- Členem struktury může být jakýkoli datový typ
- Včetně polí, výčtových a strukturovaných datových typů:

```
typedef struct {
    char jmeno[20];
    datum narozen;
} osoba;
osoba teta = { "Eva", { 12, 3, 1989 } };
teta.jmeno[1] = 'm';
teta.narozen.den = 11;
```

- Členem struktury **nemůže** být ta samá struktura:

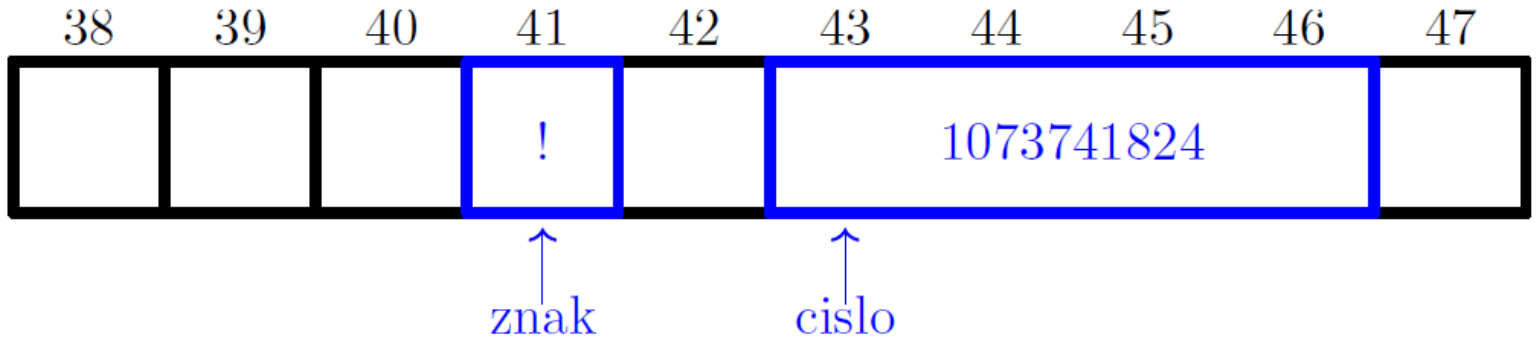
```
typedef struct {
    char jmeno[20]; osoba partner;
} osoba;
```

- Lze obejít použitím ukazatelů

Ukazatele

- Víme, že proměnné jsou za běhu programu uloženy v paměti
- Zatím nás ale nezajímalo kde
- Ukazatele = proměnné, které obsahují adresu (jiné proměnné)
- V literatuře se vyskytuje i anglický ekvivalent – pointery
- Příklad:

```
char znak = '!';
int cislo = 1073741824;
```



Vytvoření ukazatele

- Ukazatel je potřeba před použitím definovat
- V jazyku C rozlišujeme ukazatele pro práci s různými datovými typy (např. ukazatel na int, ukazatel na double, ukazatel na ukazatel na char)
- Typ ukazatele určuje, s jakými typy hodnot umí ukazatel pracovat
- Ukazatele jsou obvykle uloženy na 4/8 bytech (což není v ilustracích v této prezentaci dodržováno)
- Definice obecně:
  *typ *identifikátor = inicializační_adresa;*
- Příklady:

```
char *adresa_znaku;
double *adresa_cisla;
long *adresa_celeho_cisla;
int **adresa_adresy_cisla;
char *retezec = "Ahoj svete";
```

Práce s ukazateli

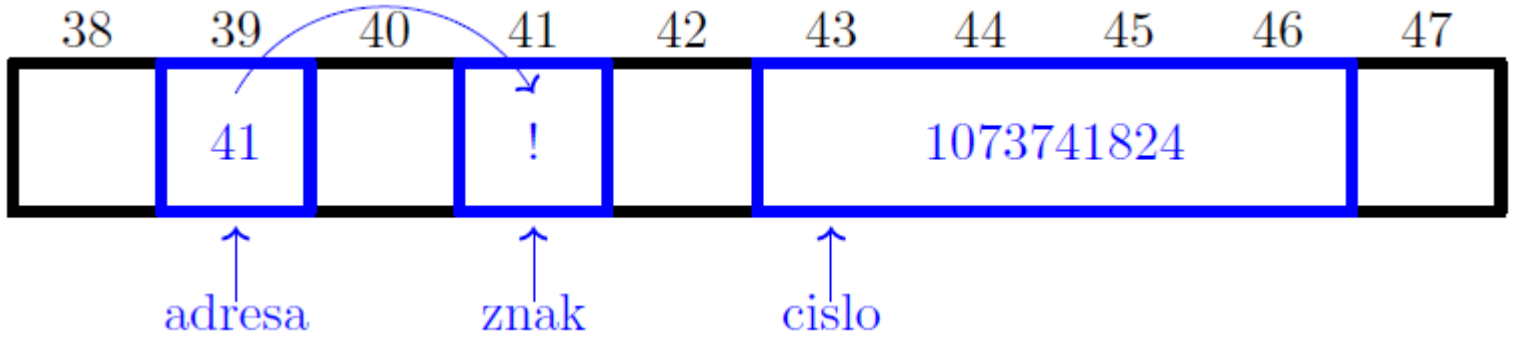
- Adresu proměnné zjistíme pomocí operátoru adresy &

```
char znak = '!';
int cislo = 1073741824;
char *adresa = &znak;
scanf("%i", &cislo);
```

- Hodnotu na dané adrese zjistíme pomocí operátoru dereference *

```
printf("%c %c\n", znak, *adresa);
```

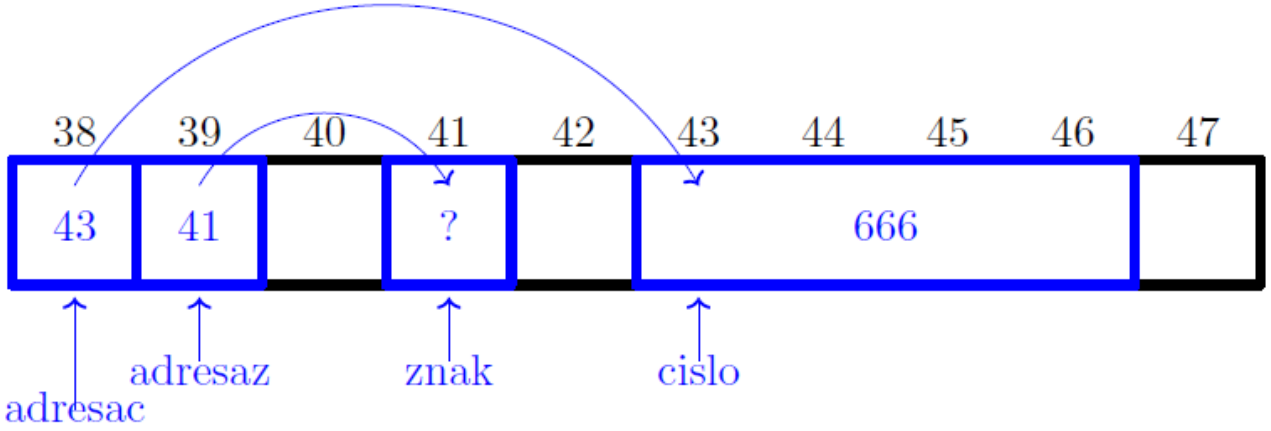
- Správně to funguje pouze při použití odpovídajícího typu ukazatele



Dereference a typ ukazatele

- Dereference získá obsah paměti od dané adresy
- Jejíž velikost odpovídá typu ukazatele
- Příklady:

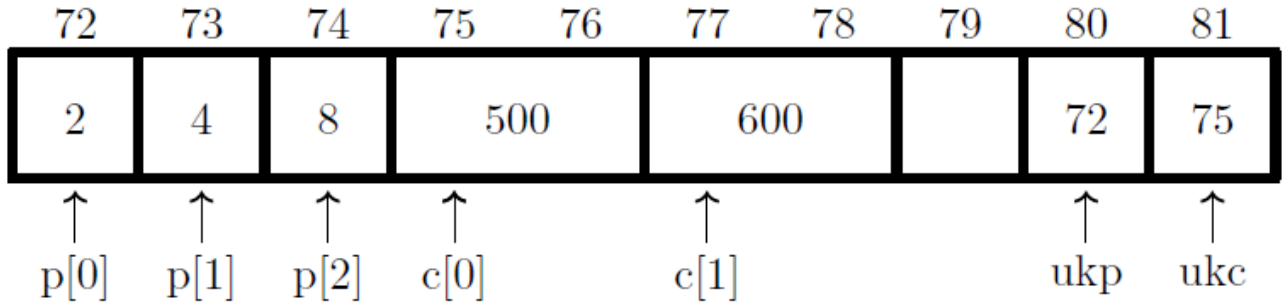
```
char* adresaz = &znak;
*adresaz = '?';
int* adresac;
adresac = &cislo;
*adresac = 666;
printf("%c, %i\n", znak, cislo);
```



Aritmetika ukazatelů

- S ukazateli lze provádět pouze některé aritmetické operace:
  - Přičtení nebo odečtení celého čísla
  - Rozdíl dvou ukazatelů (stejného typu)
- Příklady:

```
char p[] = { 2, 4, 8 }; short c[] = { 500, 600 };
char *ukp = &p[0]; short *ukc = &c[0];
printf("%i\n", *(ukp + 1)); // 4
printf("%i\n", *(ukc + 1)); // 600, deref. ukc+(1*sizeof(short))
```



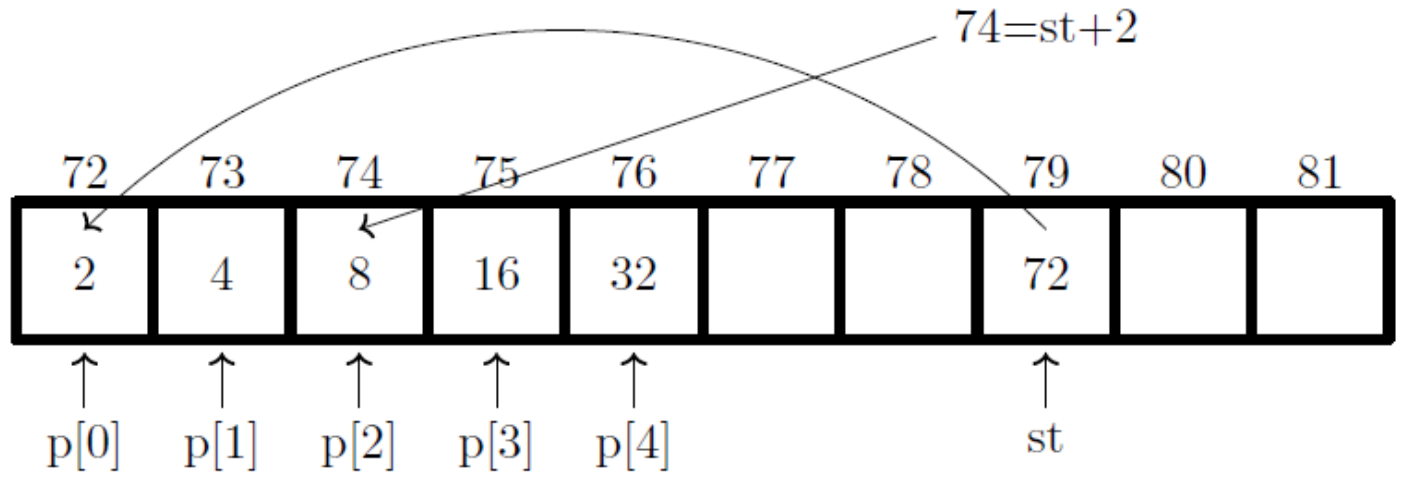
Aritmetika ukazatelů – příklad

```
int data[] = { 10, 20, 15, 30, 45 , 0};
int hledane = 30;
int *pom;

pom = &data[0]; // lze zjednodusit – viz dale
while (*pom) {
    printf("%i, ", *pom);
    if (*pom == hledane) break;
    pom++;
}

printf("Nalezeno %d na indexu %d.\n", *pom, pom - &data[0]);
```

Ukazatele a pole

- Pole je konstantní ukazatel na svůj první prvek
- S poli a ukazateli na paměť zaplněnou daty daného typu se pracuje téměř stejně
- Příklad:

```
char p[] = { 2, 4, 8, 16, 32 };
char *st = p;
printf("%i, %i, %i\n", p[2], *(p + 2), *(st + 2));
// Vypise: 8, 8, 8
```



Ukazatele a pole – příklad

- Práce s polem s použitím a bez použití operátoru indexu

```
char str[] = "Ahoj Svete!";
char *pom;

for (int i = 0; i<(int)strlen(str); i++)
    printf("%c, ", str[i]);

for (int i = 0; i<(int)strlen(str); i++)
    printf("%c, ", *(str+i));

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

Ukazatele a struktury

- Pro přístup ke členu struktury pomocí ukazatele na strukturu používáme ->

```
typedef struct { char jmeno[20]; int vek; } osoba;
osoba lide[5];
(*(lide + 2)).vek = 5;
(lide + 2)->vek = 5;
```

- Vytvoření struktury, která má jako člen „sama sebe“

```
typedef struct os{
    char jmeno[20];
    struct os *partner;
} osoba;

osoba zamestnanec = { "Adam", NULL };
osoba manzelka = { "Eva", NULL };
manzelka.partner = &zamestnanec;
manzelka.partner->partner = &manzelka;
```