Štruktúra záznam
Operácie s bitovými údajmi

1. Štruktúra záznam
   - zložený typ štruktúry záznam
   - varianty štruktúr záznam
   - reprezentácia štruktúry záznam
   - použitie štruktúry záznam v jazyku C
2. Operácie s bitovými údajmi
   - operácie s bitovými údajmi v jazyku C
   - operátory a výrazy s bitovými údajmi
   - bitové polia

Štruktúrované údajové objekty

- Najbežnejší spôsob vytvárania štruktúrovaných typov
- **Spájanie ľubovoľných prvkov (zložiek) do zložených typov**
- Príklady
  - komplexné čísla zložené z dvoch reálnych čísel
  - súradnice bodov zložené z dvoch alebo viacerých reálnych čísel
  - opis osoby pomocou niekoľkých charakteristík (meno, dátum narodenia, pohlavie, atď.)

Zložený typ

- Zložený typ je **kartziánsky súčin** jeho zložkových typov
- **Množina hodnôt** zloženého typu je tvorená všetkými možnými kombináciami hodnôt jednotlivých typov zložiek
- **Kardinalita** zloženého typu je daná súčinom kardinalít typov zložiek

Štruktúra záznam

- V oblasti spracovania údajov sa zložené typy používajú na štruktúrovaný opis osôb alebo predmetov
  - zaznamenávajú podstatné charakteristiky osôb alebo predmetov
- Najčastejšie sa tieto údaje vyskytujú ako záznamy v súboroch alebo databázach
- Pri opise zložitých štruktúr takejto povahy uprednostňujeme termín **záznam** pred pojmom karteziánsky súčin

Definícia štruktúry záznam

- Definícia štruktúry záznam (zjednodušene štruktúry) v jazyku C

```
struct [názov] {
  typ1 zložka1;
  typ2 zložka2;
  ...
  typn zložkan;
} [zoznam_premenných];
```

- Ak je definovaný názov štruktúry, môžeme ho použiť ako nový typ na definíciu ďalších premenných

```
struct názov zoznam_premenných;
```


Príklady použitia

- Reprezentácia súradníc bodu štruktúrou

```
struct bod {
  double x;
  double y;
};
```

- Definícia konkrétnych bodov pomocou štruktúry

```
struct bod a;
struct bod b = {10, 20};
```

- Reprezentácia osoby štruktúrami

```
struct datum { int den, mesiac, rok };
struct osoba {
  char meno[30];
  struct datum datum_narodenia;
  unsigned long evidencne_cislo;
};
```

- Definícia konkrétnych osôb a dátumov pomocou štruktúr

```
struct osoba zamestnavatel, zamestnanci[20], *p;
struct datum dnes, splatnost;
```

Výber zložiek štruktúry záznam

- Identifikátory zložiek záznamov sú uvedené v rámci definície zloženého typu
- Prístup ku zložkám je možný pomocou operátora sprístupnenia zložky záznamu (**selektora**) aplikovaného na premenné typu záznam

  `premenná.zložka`$_i$

- Príklady

```
a.x
zamestnavatel.meno
zamestnanec[i].datum_narodenia.mesiac
dnes.den
```


Výber zložiek štruktúry záznam

- Ak je premenná typu záznam definovaná ako **smerník na štruktúru záznam**, môžeme použiť namiesto

  `(*smerník).zložka`$_i$

  iný operátor výberu zložky záznamu pre jednoduchší zápis

  `smerník->zložka`$_i$

- Príklad

```
p->meno
```

Vlastnosti štruktúr záznam

- Ľubovoľný prístup k jednotlivým zložkám, podobne aj v poli
- Všeobecnejšia ako pole, pretože jednotlivé zložky nemusia byť rovnakého typu
- Selektory zložiek záznamov sú pevne dané identifikátormi v rámci definície záznamu
  - pri použití poľa môžu byť selektory (indexy) vypočítateľné hodnoty, čo je flexibilnejší prístup
- So štruktúrami pracujeme ako s celkom, môžeme ich kopírovať resp. priraďovať ako celok
- Môžeme získať adresu štruktúry a pracovať so smerníkom na štruktúru
  - pri použití štruktúr ako parametrov funkcií si treba uvedomiť, že sa odovzdávajú ako jedna komplexná hodnota, ktorá sa musí celá prekopírovať
  - je efektívnejšie odovzdávať smerníky na štruktúry, to isté platí o odovzdávaní štruktúry ako návratovej hodnoty funkcie

Varianty štruktúr záznam

- Z praktického hľadiska je výhodné považovať dva typy za varianty toho istého typu
- Napr. typ súradnice môžeme považovať za zjednotenie dvoch variantov – kartéziánskej (dve dĺžky) a polárnej (jedna dĺžka, jeden uhol) súradnicovej sústavy
- Takáto definícia štruktúry záznam umožňuje, aby údajový objekt bol **interpretovaný rôznym spôsobom**
- Kardinalita takejto štruktúry je daná súčtom kardinalít jednotlivých typov variantov

Definícia štruktúry záznam s variantmi

- Definícia štruktúry záznam s variantmi (zjednodušene union) v jazyku C

```
union [názov] {
  typ1 zložka1;
  typ2 zložka2;
  ...
  typn zložkan;
} [zoznam_premenných];
```

- Takto definovaný union umožňuje prístup k jednému údaju pomocou rôznych *n* zložiek rôznych typov
- **V jazyku C je práca s unionom je rovnaká ako práca so štruktúrou**
  - treba si uvedomiť, že všetky zložky unionu uchovávajú ten istý údaj a nie *n* rôznych údajov
  - prístup ku konkrétnej zložke unionu závisí od toho, akým spôsobom chceme s údajom pracovať (máme *n* rôznych spôsobov)

Príklady použitia

- Pri používaní štruktúr záznam s variantmi často kombinujeme použitie štruktúr a unionov
- Pre identifikáciu konkrétnej varianty často zavádzame zložku nazývanú **diskriminátor typu** alebo **rozlišovacia zložka**

- Definícia bodu pomocou štruktúry s dvoma variantmi (karteziánske alebo polárne súradnice)

```
struct bod {
  enum {KARTEZIANSKE, POLARNE} druh;
  union {
    struct {double x, y;} kartezianske;
    struct {double r, fi;} polarne;
  } suradnice;
};

struct bod a;
```

Výber variantov zložiek štruktúry

- Selektor pre výber jednotlivých zložiek takéhoto záznamu je daný úrovňami definície jednotlivých štruktúr a unionov

```
a.druh
a.suradnice.kartezianske.x
a.suradnice.polarne.fi
```

- Vo všeobecnosti môže programátor použiť ktorúkoľvek variantu záznamu `suradnice`, čo môže spôsobiť vážne sémantické chyby programu
- Odporúča sa, aby príslušné operácie s jednotlivými variantmi boli striktne rozlíšené (použitie vetvenia) pomocou rozlišovacej zložky `druh`
  - takto môžeme v programe jednoducho overiť správnosť použitia selektora
  - stačí overiť, či každá vetva obsahuje iba korektné selektory

```
switch (a.druh) {
  case KARTEZIANSKE:
    printf("Kartézske: x = %g, y = %g\n",
           a.suradnice.kartezianske.x, a.suradnice.kartezianske.y);
    break;
  case POLARNE:
    printf("Polárne: r = %g, φ = %g\n",
           a.suradnice.polarne.r, a.suradnice.polarne.fi);
    break;
}
```


Reprezentácia štruktúr záznam

- Záznamy sú zobrazované do pamäti postupným zobrazovaním ich jednotlivých zložiek
- Adresu *i*-tej zložky záznamu vzhľadom na začiatočnú adresu záznamu nazývame **posunutie** (alebo **offset**) a_i zložky i

$$a_i = s_1 + s_2 + ... + s_{i-1}$$

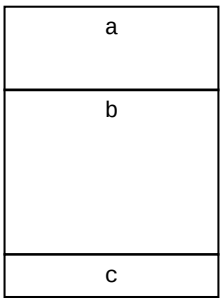
pričom s_j je veľkosť j-tej zložky záznamu

- Celková veľkosť záznamu je daná súčtom veľkostí jej zložiek
- Zložky záznamu sú prístupné len prostredníctvom presne stanovených identifikátorov
  - každé s_j môže mať inú hodnotu a výpočet posunutia sa preto nedá automaticky vypočítať z pozície zložky
- Posunutia zložiek sú známe už počas kompilácie
  - nie je potrebné vykonávať výpočet pozície zložky (ako napr. v prípade indexov poľa) počas realizácie programu

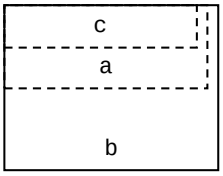
Reprezentácia štruktúr záznam s variantmi

- Štruktúra záznam s variantmi (union) reprezentuje jednu informáciu prístupnú viacerými spôsobmi
- Z toho vyplýva, že každá zložka takéhoto unionu je **zobrazená na začiatočnej adrese** celého záznamu
- Celková veľkosť unionu je daná **veľkosťou najväčšej zložky unionu**

```
struct {
  int a;
  float b;
  char c;
};
```




```
union {
  int a;
  float b;
  char c;
};
```



Operácie s bitovými údajmi

- Tvorba výrazov pre vykonávanie základných operácií s **jednotlivými bitmi alebo segmentmi bitov** (napr. bajtmi alebo bitovými poliami)
- Prostriedok pre prístup k údajom na **najnižšej – strojovej úrovni**
- Používajú sa na manipuláciu bitov v rámci údajových objektov **celočíselných typov** (`char`, `int`, `long`, `unsigned`)

Použitie operácií s bitmi

- Programovanie ovládačov (driver) rozličných technických zariadení a aplikácií s priamym prístupom k nim
- Pamäťovo úsporná implementácia logických údajových objektov (stačí 1 bit) a údajov, ktoré majú užší interval hodnôt, ako základné typy (napr. pre zakódovanie 8 hodnôt nám stačia iba 3 bity)
- Programovanie emulátorov operácií v pohyblivej rádovej čiarke, ak nie je k dispozícii príslušný procesor

Bitové operátory

Operátor	Význam	Priorita	Asociatívnosť
~	jednotkový doplnok bitov v slove	14	←
>>	posun bitov v slove vpravo	11	→
<<	posun bitov v slove vľavo	11	→
&	logický súčin bitov dvoch slov	8	→
^	neekvivalencia bitov dvoch slov	7	→
|	logický súčet bitov dvoch slov	6	→

Jednotkový doplnok bitov v slove

- Táto operácia vykoná inverziu všetkých bitov v slove – bit s hodnotou 1 zmení hodnotu na 0 a bit s hodnotou 0 zmení hodnotu na 1

```
~slovo
```

- Používa sa tiež na vytváranie implementačne nezávislých konštánt
  - slovo typu `int` môže byť na rôznych počítačových systémoch implementované rôzne (napr. 16 bitové slovo alebo 32 bitové slovo)

0177777	šestnásťbitové slovo so všetkými bitmi hodnoty 1
~0	slovo so všetkými bitmi hodnoty 1 bez ohľadu na koľkých bitoch je implementované

Posun bitov v slove

- Bity v slove sú posúvané o konkrétny počet v slove doprava alebo doľava

```
slovo << n
slovo >> n
```

- Posun o *n* bitov doľava predstavuje násobenie slova číslom 2^n
- Posun o *n* bitov doprava predstavuje delenie slova číslom 2^n
- Pri posune doľava sa sprava do slova nasúvajú bity s hodnotou 0
- Pri posune doprava sa zľava do slova nasúva
  - znamienkový bit – **aritmetický posun**
  - bit s hodnotou 0 – **logický posun**
  - je to implementačne závislé, pričom pre nezávislú implementáciu je potrebné definovať slovo typu `unsigned` (potom sa vždy realizuje logický posun)

Logický súčin bitov dvoch slov

- Táto operácia vykoná logický súčin jednotlivých bitov na tých istých pozíciách dvoch slov

  ```
  slovo1 & slovo2
  ```

- Používa sa často na nulovanie bitov v bitovom segmente

  ```
  int a;
  a = a & 0177400;
  ```

  spôsobí vynulovanie nižších 8 bitov v premennej `a`, za predpokladu, že typ `int` je realizovaný na 16 bitoch

  lepšie je použiť implementačne nezávislú konštantu

  ```
  a = a & ~0377;
  ```


Logický súčet bitov dvoch slov

- Táto operácia vykoná logický súčet jednotlivých bitov na tých istých pozíciách dvoch slov

  $slovo_1$ | $slovo_2$

- Používa sa často na nastavenie bitov v bitovom segmente

  ```
  int a, b;
  b = b << 4;
  a = a & ~0360;
  a = a | b;
  ```

  zápis celočíselnej hodnoty z premennej `b` do premennej `a` do bitového segmentu od 4. po 7. bit (v premennej `b` predpokladáme hodnotu 0 – 15)

Bitové polia

- Umožňujú **štruktúrovaný pohľad na neštruktúrovaný typ** `int`
- Definujú (deklarujú) sa ako **jedna zložka štruktúry záznam**
- Umožňujú odkazovať sa na zložky, veľkosť ktorých je **špecifikovaná v bitoch**
- Vhodné ak chceme **považovať údajový objekt za súbor bitov** (napr. kvôli šetreniu pamäťového miesta, pri komunikácii s V/V zariadeniami, atď.)

Definícia bitového poľa

- Definícia bitového poľa je v jazyku C realizovaná pomocou štruktúry

```
struct [názov] {
  unsigned [zložka1] : veľkosť1;
  unsigned [zložka2] : veľkosť2;
  ...
  unsigned [zložkan] : veľkosťn;
} [zoznam_premenných];
```

- Hodnota `veľkosťi` nesmie presiahnúť veľkosť typu `unsigned int`
- Ak je `veľkosťi` rovná **0**, potom ďalšia zložka bitového poľa bude začínať **na novom slove** `unsigned int`
- Ak je **vynechaný identifikátor** `zložkai`, potom bude ďalšie **bitové pole posunuté** o veľkosť uvedenú za **:**

Operácia s bitovým poľom

- S bitovým poľom sú dovolené všetky operácie ako s typom `int`, okrem operátora výpočtu adresy &
- V bitovom poli nie je dovolené použiť ako prvok pole
- Bitové polia sú **implementačne závislé**

Príklady použitia bitového poľa

- Príklad použitia bitových polí ako komunikačného rozhrania nejakého zariadenia

```
struct device_interface {
  unsigned R_W : 1;  // bitové pole o veľkosti 1 bit
  unsigned : 0;  // prechod na nové slovo
  unsigned Dir1 : 1;
  unsigned Dir2 : 1;
  unsigned Dir3 : 1;
  unsigned : 4;  // 4 bity budú nevyužité
  unsigned Mode : 3;
} device;  // definícia premennej pozostávajúcej z bitových polí
```

- Bitové polia pre prácu v pohyblivej rádovej čiarke (predpokladáme `int` aj `float` na 32 bitov) a union s variantami použitia reálneho čísla

```
struct PHRC {
  unsigned znamienko_mantisy : 1;
  unsigned exponent : 8;
  unsigned mantisa : 23;
};
union {
  float x;
  struct PHRC y;
} realne_cislo;
```