Práce s bity

Programovací techniky

doc. Ing. Jiří Rybička, Dr.
Ústav informatiky
PEF MENDELU v Brně
rybicka@mendelu.cz

Celočíselné datové typy – shrnutí

Celočíselné datové typy – shrnutí

- **Přehled**

Celočíselné datové typy – shrnutí

- **Přehled**
  - char

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %
  - porovnání

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %
  - porovnání
  - **práce s bity**

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %
  - porovnání
  - **práce s bity**
- **Vztah k ostatním typům**

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %
  - porovnání
  - **práce s bity**
- **Vztah k ostatním typům**
  - kompatibilní s bool

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %
  - porovnání
  - **práce s bity**
- **Vztah k ostatním typům**
  - kompatibilní s bool
  - kompatibilní se znakovou hodnotou

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %
  - porovnání
  - **práce s bity**
- **Vztah k ostatním typům**
  - kompatibilní s bool
  - kompatibilní se znakovou hodnotou
  - automatické konverze navzájem

Celočíselné datové typy – shrnutí

- **Přehled**
  - char
  - int
  - short int
  - long int
  - long long int
  - signed, unsigned
- **Operace**
  - aritmetické: +, -, *, /, %
  - porovnání
  - **práce s bity**
- **Vztah k ostatním typům**
  - kompatibilní s bool
  - kompatibilní se znakovou hodnotou
  - automatické konverze navzájem
  - automatické konverze s textovými soubory

Práce s bity

- **Motivace**

Práce s bity

- **Motivace**
  - uložení více logických hodnot do jednoho celku

Práce s bity

- **Motivace**
  - uložení více logických hodnot do jednoho celku
  - indikace určité množiny hodnot

Práce s bity

- **Motivace**
  - uložení více logických hodnot do jednoho celku
  - indikace určité množiny hodnot
  - náhrada aritmetických operací

Práce s bity

- **Motivace**
  - uložení více logických hodnot do jednoho celku
  - indikace určité množiny hodnot
  - náhrada aritmetických operací
  - možnosti zobrazení dat uložených v paměti

Práce s bity

- **Motivace**
  - uložení více logických hodnot do jednoho celku
  - indikace určité množiny hodnot
  - náhrada aritmetických operací
  - možnosti zobrazení dat uložených v paměti
  - zabezpečení

Práce s bity

- **Motivace**
  - uložení více logických hodnot do jednoho celku
  - indikace určité množiny hodnot
  - náhrada aritmetických operací
  - možnosti zobrazení dat uložených v paměti
  - zabezpečení
  - komprimace

Práce s bity

- **Motivace**
  - uložení více logických hodnot do jednoho celku
  - indikace určité množiny hodnot
  - náhrada aritmetických operací
  - možnosti zobrazení dat uložených v paměti
  - zabezpečení
  - komprimace
  - šifrování

Operace s bity

- Velmi názorné je uvádět konstantní hodnoty šestnáctkově – převod do binární podoby je zcela přímočarý (jedna šestnáctková číslice představuje právě 4 binární číslice)

Operace s bity

- Velmi názorné je uvádět konstantní hodnoty šestnáctkově – převod do binární podoby je zcela přímočarý (jedna šestnáctková číslice představuje právě 4 binární číslice)
- Bitový logický součin `&`

```
    0000 1100
 &  0001 1010
-------------
    0000 1000
```

zapsáno ve zdrojovém textu: `0x0c & 0x1a = 0x08`

Operace s bity

- Velmi názorné je uvádět konstantní hodnoty šestnáctkově – převod do binární podoby je zcela přímočarý (jedna šestnáctková číslice představuje právě 4 binární číslice)
- Bitový logický součin &

```
    0000 1100
 &  0001 1010
-------------
    0000 1000
```

zapsáno ve zdrojovém textu: `0x0c & 0x1a = 0x08`

- Bitový logický součet |

```
    0000 1100
 |  0001 1010
-------------
    0001 1110
```

zapsáno ve zdrojovém textu: `0x0c | 0x1a = 0x1e`

Operace s bity

- Bitový výhradní součet ^

```
     0000 1100
 ^   0001 1010
-------------
     0001 0110
```

zapsáno ve zdrojovém textu: `0x0c ^ 0x1a = 0x16`

Operace s bity

- Bitový výhradní součet ^

```
     0000 1100
 ^   0001 1010
-------------
     0001 0110
```

zapsáno ve zdrojovém textu: `0x0c ^ 0x1a = 0x16`

- Bitová negace ~

```
 ~   0001 1010
-------------
     1110 0101
```

zapsáno ve zdrojovém textu: `~0x1a = 0xe5`

Operace s bity

- Bitový výhradní součet ^

```
    0000 1100
 ^  0001 1010
-------------
    0001 0110
```

zapsáno ve zdrojovém textu: `0x0c ^ 0x1a = 0x16`

- Bitová negace ~

```
 ~  0001 1010
-------------
    1110 0101
```

zapsáno ve zdrojovém textu: `~0x1a = 0xe5`

- Bitový posuv vlevo <<

`0x25 << 2 = 0x94`

`0010 0101` $\longrightarrow$ `1001 0100`

Posuv vlevo o *n* bitů znamená násobení hodnotou 2^n.

Operace s bity

- Bitový posuv vpravo >>

  `0x31 >> 2 = 0x0c`

  `0011 0001` ⟶ `0000 1100`

  Posuv vpravo o *n* bitů znamená dělení hodnotou 2^n.

  POZOR! Logický posuv × aritmetický posuv:

  Co se zobrazí na výstupu?

```
signed char A=-128;  //A=1000 0000
unsigned char B=128; //B=1000 0000
A=A>>1;  B=B>>1;
cout << int(A) << endl //A=1100 0000
     << int(B) << endl;//B=0100 0000
```

  Na výstupu bude -64 a 64.

Bitová maska

- Bitová maska je posloupnost bitů, kde hodnota 0 potlačuje hodnotu, hodnota 1 ponechává hodnotu.

Bitová maska

- Bitová maska je posloupnost bitů, kde hodnota 0 potlačuje hodnotu, hodnota 1 ponechává hodnotu.
- Maska se logicky vynásobí s příslušnou hodnotou. Výsledkem jsou ponechané bity původní hodnoty.

Bitová maska

- Bitová maska je posloupnost bitů, kde hodnota 0 potlačuje hodnotu, hodnota 1 ponechává hodnotu.
- Maska se logicky vynásobí s příslušnou hodnotou. Výsledkem jsou ponechané bity původní hodnoty.
- Příklad: Maska `0010 0010` ponechá 2. a 6. bit původní hodnoty, ostatní bity budou nulové: `xx1x xx0x & 0010 0010 = 0010 0000`

Bitová maska

- Bitová maska je posloupnost bitů, kde hodnota 0 potlačuje hodnotu, hodnota 1 ponechává hodnotu.
- Maska se logicky vynásobí s příslušnou hodnotou. Výsledkem jsou ponechané bity původní hodnoty.
- Příklad: Maska `0010 0010` ponechá 2. a 6. bit původní hodnoty, ostatní bity budou nulové: `xx1x xx0x & 0010 0010 = 0010 0000`
- Jednoduše tak zjistíme hodnotu libovolného bitu, např.: Jakou hodnotu má 4. bit v celočíselné proměnné `Arg`?

```
int Arg;
cin >> Arg;
if (Arg & 8 > 0) cout << ”bit je 1” << endl;
    else cout << ”bit je 0” << endl;
```

Nastavení/nulování bitů

- V proměnné můžeme libovolně ovládat každý bit bez ohledu na hodnoty ostatních bitů. Použijeme k tomu vhodnou masku a vhodnou operaci.

Nastavení/nulování bitů

- V proměnné můžeme libovolně ovládat každý bit bez ohledu na hodnoty ostatních bitů. Použijeme k tomu vhodnou masku a vhodnou operaci.
- Nastavení *n*-tého bitu na hodnotu 1:

```
hodnota | maska
```

Nastavení/nulování bitů

- V proměnné můžeme libovolně ovládat každý bit bez ohledu na hodnoty ostatních bitů. Použijeme k tomu vhodnou masku a vhodnou operaci.
- Nastavení *n*-tého bitu na hodnotu 1:

  `hodnota | maska`
- Nulování *n*-tého bitu: `hodnota & maska`

Nastavení/nulování bitů

- V proměnné můžeme libovolně ovládat každý bit bez ohledu na hodnoty ostatních bitů. Použijeme k tomu vhodnou masku a vhodnou operaci.
- Nastavení *n*-tého bitu na hodnotu 1:

  `hodnota | maska`
- Nulování *n*-tého bitu: `hodnota & maska`
- Příklady:

Nastavení/nulování bitů

- V proměnné můžeme libovolně ovládat každý bit bez ohledu na hodnoty ostatních bitů. Použijeme k tomu vhodnou masku a vhodnou operaci.
- Nastavení *n*-tého bitu na hodnotu 1:

  `hodnota | maska`
- Nulování *n*-tého bitu: `hodnota & maska`
- Příklady:
  - Nastavte 7. bit proměnné `Prop` na 1:

    ```
    Prop = Prop | 0x40;
    ```

Nastavení/nulování bitů

- V proměnné můžeme libovolně ovládat každý bit bez ohledu na hodnoty ostatních bitů. Použijeme k tomu vhodnou masku a vhodnou operaci.
- Nastavení *n*-tého bitu na hodnotu 1:

  `hodnota | maska`
- Nulování *n*-tého bitu: `hodnota & maska`
- Příklady:
  - Nastavte 7. bit proměnné `Prop` na 1:

    ```
    Prop = Prop | 0x40;
    ```
  - Vynulujte 6. bit proměnné `Prop`:

    ```
    Prop = Prop & 0xdf;
    ```

Nastavení/nulování bitů

- V proměnné můžeme libovolně ovládat každý bit bez ohledu na hodnoty ostatních bitů. Použijeme k tomu vhodnou masku a vhodnou operaci.
- Nastavení *n*-tého bitu na hodnotu 1:

  `hodnota | maska`
- Nulování *n*-tého bitu: `hodnota & maska`
- Příklady:
  - Nastavte 7. bit proměnné `Prop` na 1:

    ```
    Prop = Prop | 0x40;
    ```
  - Vynulujte 6. bit proměnné `Prop`:

    ```
    Prop = Prop & 0xdf;
    ```
  - Uložte do proměnné `Vrch` hodnotu horního půlbajtu proměnné `Kod`:

    ```
    Vrch = (Kod & 0xf0) >> 4;
    ```

Zjišťování hodnot význačných bitů

- Význačný bit: nejnižší, nejvyšší

Zjišťování hodnot význačných bitů

- Význačný bit: nejnižší, nejvyšší
- Využívá se povaha binárních čísel a použité kódování

Zjišťování hodnot význačných bitů

- Význačný bit: nejnižší, nejvyšší
- Využívá se povaha binárních čísel a použité kódování
- Zjištění hodnoty nejnižšího bitu celočíselné proměnné `P`:
  liché binární číslo má nejnižší bit roven 1: `P % 2`

Zjišťování hodnot význačných bitů

- Význačný bit: nejnižší, nejvyšší
- Využívá se povaha binárních čísel a použité kódování
- Zjištění hodnoty nejnižšího bitu celočíselné proměnné `P`:
  liché binární číslo má nejnižší bit roven 1: `P % 2`
- Zjištění hodnoty nejvyššího bitu libovolného znaménkového celočíselného typu:
  záporná čísla v doplňkovém kódu mají nejvyšší bit roven 1: `P < 0`

Zjišťování hodnot význačných bitů

- Význačný bit: nejnižší, nejvyšší
- Využívá se povaha binárních čísel a použité kódování
- Zjištění hodnoty nejnižšího bitu celočíselné proměnné `P`:
  liché binární číslo má nejnižší bit roven 1: `P % 2`
- Zjištění hodnoty nejvyššího bitu libovolného znaménkového celočíselného typu:
  záporná čísla v doplňkovém kódu mají nejvyšší bit roven 1: `P < 0`
- Řídicí znak v ASCII má binární prefix `000`

Zjišťování hodnot význačných bitů

- Význačný bit: nejnižší, nejvyšší
- Využívá se povaha binárních čísel a použité kódování
- Zjištění hodnoty nejnižšího bitu celočíselné proměnné `P`:
  liché binární číslo má nejnižší bit roven 1: `P % 2`
- Zjištění hodnoty nejvyššího bitu libovolného znaménkového celočíselného typu:
  záporná čísla v doplňkovém kódu mají nejvyšší bit roven 1: `P < 0`
- Řídicí znak v ASCII má binární prefix `000`
- Číslice v ASCII má binární prefix `0011`

Zjišťování hodnot význačných bitů

- Význačný bit: nejnižší, nejvyšší
- Využívá se povaha binárních čísel a použité kódování
- Zjištění hodnoty nejnižšího bitu celočíselné proměnné `P`:
  liché binární číslo má nejnižší bit roven 1: `P % 2`
- Zjištění hodnoty nejvyššího bitu libovolného znaménkového celočíselného typu:
  záporná čísla v doplňkovém kódu mají nejvyšší bit roven 1: `P < 0`
- Řídicí znak v ASCII má binární prefix `000`
- Číslice v ASCII má binární prefix `0011`
- Binární prefixy UTF-8 znaků (viz dále)

Příklady

- Výpis hodnoty proměnné `Cislo` binárně (nejnižší bit je vypsán jako první):

Příklady

- Výpis hodnoty proměnné `Cislo` binárně (nejnižší bit je vypsán jako první):
- ```
  char Cislo;
  cin >> Cislo;
  for (int i=0; i<7; i++) {
     cout << Cislo % 2;
     Cislo = Cislo >> 1;
  }
  ```
  ```

Příklady

- Výpis hodnoty proměnné `Cislo` binárně (nejnižší bit je vypsán jako první):
- ```
  char Cislo;
  cin >> Cislo;
  for (int i=0; i<7; i++) {
     cout << Cislo % 2;
     Cislo = Cislo >> 1;
  }
  ```
- Zjištění počtu binárních jedniček v jednom bajtu:

```
char Cislo;  int pocet=0;
cin >> Cislo;
while (Cislo>0) {
   pocet += Cislo & 1;
   Cislo = Cislo >> 1;
}
```
```

Příklady

- Zjistěte, kolik bajtů má znak v kódování UTF-8 uložený v řetězci. Jednobajtový znak má v nejvyšším bitu 0, dvoubajtový znak začíná bajtem, jehož binární prefix je `110`, tříbajtový `1110` a čtyřbajtový `11110`.

Příklady

- Zjistěte, kolik bajtů má znak v kódování UTF-8 uložený v řetězci. Jednobajtový znak má v nejvyšším bitu 0, dvoubajtový znak začíná bajtem, jehož binární prefix je `110`, tříbajtový `1110` a čtyřbajtový `11110`.
- ```
  char retez[10];
  cin >> retez;  //přečtení řetězce ze vstupu
  unsigned char Z=retez[0]; //první bajt
  if ((Z & 0x80) == 0) cout << ”1 B” << endl;
  if ((Z >> 5) == 6)   cout << ”2 B” << endl;
  if ((Z & 0xe0) == 0xe0)
      cout << ”3 B” << endl;
  if (((Z & 0xf8) >> 3) == 30)
      cout << ”4 B” << endl;
  ```
  ```

Příklady

- Přečtěte textový soubor ”clanek.txt”a zašifrujte ho znakem získaným ze standardního vstupu. Výsledek vypište do binárního souboru ”sifra.dat”.

Příklady

- Přečtěte textový soubor ”clanek.txt”a zašifrujte ho znakem získaným ze standardního vstupu. Výsledek vypište do binárního souboru ”sifra.dat”.
- ```
  #include <iostream>
  #include <fstream>
  using namespace std;

  int main(){
    ifstream vstup (”clanek.txt”);
    ofstream vystup (”sifra.dat”, ios::binary);
    char Z, heslo;
    cin >> heslo;
  ```
  ```

Příklady

- (pokračování)

```
  while (not vstup.eof()){
    Z = vstup.get(); //čtení znak po znaku
    Z = Z ^ heslo;   //šifra
    if (not vstup.eof()) vystup.put(Z);
         //výstup, není-li to konec souboru
  }
  vystup.close(); //uzavření výstupu
  return 0;
}
```