Pracovní list 4: Práce s bity

Co už máme znát

- přehled celočíselných datových typů a práce s nimi;
- způsob zobrazení celých čísel v doplňkovém kódu;
- způsob zobrazení čísel s plouvoucí řádovou čárkou;
- způsob zobrazení hodnot v kódování BCD;
- znakový kód ASCII;
- znakový kód UTF-8;
- zabezpečení dat příčnou a podélnou paritou;
- bitové operace: součet, součin, výhradní součet, negace.

Kontrolní otázky

4.1 Jak jsou zobrazeny znaky v kódování UTF-8?

4.2 Jak se vypočítá paritní informace příčné (podélné) parity?

4.3 Co je bitová maska a jak se používá?

4.4 Jak lze v jazyce C++ pracovat s jedním paměťovým místem dvěma různými způsoby? (struktura `union`)

4.5 Jak lze zjistit hodnoty význačných bitů (nejvyšší, nejnižší)?

4.6 Jakou vlastnost operace XOR využíváme při jednoduchém šifrování?

Příprava na cvičení

Ve cvičení budeme potřebovat překladač jazyka C++, editor pro přípravu zdrojových textů a vybavení příkazového řádku. Pro jednotlivé úlohy jsou k dispozici soubory s daty, případně výsledné soubory v adresáři `/home/rybicka/vyuka/progt/cecko/cviceni/cv04` na serveru akela. Konkrétní jména těchto souborů jsou uvedena u jednotlivých úloh.

Řešené příklady

Příklad 4.1 Vstup je tvořen souvislým textem. Zašifrujte jej operací XOR tříznakovým heslem složeným z ASCII znaků uloženým v souboru `heslo.txt`. Výsledek vypište do binárního souboru `sifra.cif`.

Řešení: Podstatou šifrování je získání šifrového bajtu tím, že se postupně každý znak původního textu sloučí bitovou operací XOR se znakem hesla, který je právě na řadě. Začíná se s prvním znakem hesla, když se délka hesla vyčerpá, jede se znovu od jeho prvního znaku (viz řádek 152. Postupně tedy čteme celý vstupní soubor znak po znaku, vytváříme šifrové bajty střídáním heslových znaků a vypisujeme výsledek do binárního souboru (jde samozřejmě již o binární data). Výhodou tohoto jednoduchého způsobu šifrování je snadné dešifrování zcela identickým způsobem – opět se použije stejné heslo a stejná operace. Oba procesy lze tedy řešit stejným programem. Zároveň se jedná o symetrické šifrování, heslo může být pro každý soubor jiné.

```
137 #include <iostream>
138 #include <fstream>
139 using namespace std;
140
141 int main(){
142   ifstream soubheslo ("heslo.txt");
143   ofstream soubsifra ("sifra.cif", ios::binary); //soubory
144   const int DelkaHesla = 3;
145   char heslo[DelkaHesla], Znak, Sif;
146   int indexHeslo=0;
147   if (soubheslo.is_open()) {
148      soubheslo >> heslo;  //přečtení hesla ze souboru
149      while (cin.get(Znak)) {  //čtení vstupu znak po znaku
150        Sif = Znak ^ heslo[indexHeslo]; //šifrování
151        soubsifra.put(Sif);      //výpis šifry do výstupu
152        if (indexHeslo==DelkaHesla-1) indexHeslo = 0; //heslo znovu
153           else indexHeslo++;    //následující znak hesla
154      }
155      soubsifra.close();    //uzavření výstupu
156   } else cerr << "Soubor s heslem nelze číst" << endl;
157   return 0;
158 }
```

Příklad 4.2 Na vstupu se nachází neuspořádaná posloupnost celočíselných hodnot. Zjistěte, která čísla z intervalu 0 až 31 se na vstupu nevyskytovala.

Řešení: Využijeme skutečnosti, že informace o výskytu čísla ve vstupní řadě je logická hodnota, kterých potřebujeme celkem 32 (hlídáme 32 čísel v intervalu 0 až 31). Tyto logické hodnoty „zkomprimujeme“ do jednoho čtyřbajtového celého čísla. Bit na n-té pozici tohoto prostoru bude pak ukazovat informaci o přítomnosti/nepřítomnosti příslušné hodnoty n na vstupu. Musíme jen vyřešit nastavení n-tého bitu na jedničku při přečtení čísla n z intervalu 0 až 31 a na závěr pak zjistit, které bity zůstaly nulové – tato čísla se na vstupu nevyskytovala. K nastavení na jedničku nebo přečtení informace o hodnotě bitu použijeme masky. Každá pozice bude mít svou masku a seznam všech masek si naplníme do pole `masky`. Čtené číslo bude pak indexem do pole masek a bude tedy použita odpovídající maska pro nastavení bitu. Proměnná, ve které nastavujeme jednotlivé bity, může sloužit i

jako indikace *množiny hodnot*. Každý prvek množiny je zastoupen jedním bitem, jehož pořadí udává příslušnou hodnotu. Koncept množiny (bitového pole) je využitelný v řadě aplikací.

```
159 #include <iostream>
160 #include <fstream>
161 using namespace std;
162
163 int main(){
164   unsigned long int mnozina=0; //32 bitů - na začátku vše nulové
165   unsigned long int masky[32], //pole masek
166                     m=1;
167   for (int i=0; i<32; i++) { //naplnění pole masek
168       masky[i]=m; m=m<<1;
169   }
170   unsigned int cislo;
171   while (cin>>cislo){   //čtení vstupu až do konce
172     if (cislo>=0 and cislo<=31) //kontrola intervalu
173        mnozina = mnozina | masky[cislo];  //nastavení příslušného bitu
174   }
175   for (int i=0; i<32; i++)
176     if ((mnozina & masky[i]) == 0)  //bit nulový -> číslo nebylo na vstupu
177         cout << i << endl;
178    return 0;
179 }
```

Paměťově poněkud úspornější varianta nepoužívá pole masek. Místo něj můžeme použít masku, kterou vyrobíme „na místě“ posuvem jedničky o příslušný počet míst doleva. Na řádku 173 pak můžeme psát

```
180         mnozina = mnozina | 1<<cislo;
```

a místo dvou řádků počínaje 176 zapíšeme

```
181     if ((mnozina & 1<<i) == 0) cout << i << endl;
```

Příklady

Příklad 4.3 Na vstupu se nachází neuspořádaná řada celých čísel v intervalu 0 až 63. Předpokládejte, že se čísla neopakují, každé je na vstupu nejvýše jednou. Seřaďte tato čísla pomocí modelu bitové množiny v proměnné typu `unsigned long long int`. (Jako vstup lze využít soubor `cisla.txt`.)

Příklad 4.4 Je-li množina implementována jako bitové pole, jak můžeme pak implementovat následující operace nad takovými množinami: sjednocení, průnik, doplněk?

Příklad 4.5 Na standardním vstupu se nachází text, jehož znaky jsou pouze sedmibitové ASCII. Zabezpečte tento text sudou paritou a výsledek zapište do výstupního binárního souboru `parita.dat`.

Příklad 4.6 Na disku se nachází binární soubor `ucto.xxx`. Vypočtěte bajt podélné parity všech dat tohoto souboru a vypište jeho hodnotu na výstup.

Řešení: Výsledek je 5.

Příklad 4.7 V binárním souboru `bcd.enc` se nachází posloupnost číslic kódovaných v kódu BCD. Přečtěte tento soubor a vypište na výstup tabulku četností jednotlivých číslic.

Řešení: Výsledky:

Číslice 0:	134109
Číslice 1:	68088
Číslice 2:	173083
Číslice 3:	110145
Číslice 4:	95128
Číslice 5:	105813
Číslice 6:	248621
Číslice 7:	132260
Číslice 8:	17025
Číslice 9:	45124

Příklad 4.8 Napište program, kterým zjistíte strojovou hodnotu exponentu čísla typu `double` zadaného ze vstupu.

Příklad 4.9 Napište program, kterým vypíšete binární podobu čísla typu `float` zadaného ze vstupu.

Příklad 4.10 Na standardním vstupu se nachází proud znaků nula a jednička představující obrazové informace o pixelech v monochromatickém obrazu. Napište program, který přečte a uloží tyto informace do binárního souboru tak, aby každý pixel zabíral pouze jeden bit. Pokud počet vstupních hodnot není dělitelný osmi, doplňte chybějící pixely v posledním bajtu nulami. (Jako vstup můžete využít soubor `jednanula.txt`.)

Příklad 4.11 Na standardním vstupu se nachází text časopisu v kódování UTF-8. Zjistěte, kolik obsahuje znaků. (Pro vstup lze využít soubor `data.txt`.)

Řešení: Pro soubor `data.txt` je výsledek 519 543 znaků.

Příklad 4.12 V diskovém souboru `vstup.txt` se nachází libovolný text. Zašifrujte tento text operací XOR libovolně dlouhým heslem zadaným z klávesnice a vypište výsledek do souboru `sifra.dat`. Předpokládejte, že heslo je složeno pouze ze zobrazitelných ASCII znaků.

Příklad 4.13 Dešifrujte soubor vzniklý v předchozí úloze (heslo opět vstupuje z klávesnice) a výsledek vypisujte na standardní výstup.

Řešení: Použité heslo je `cv04c++`.

Co máme po cvičení umět

- Práce se znaky v kódování UTF-8.
- Některé způsoby kódování celočíselných hodnot a jejich využití.
- Aplikace bitových operací, tvorba a aplikace masek.
- Možnost úsporného uložení informací po bitech.
- Využití informací o význačných bitech (nejvyšší, nejnižší).

Kontrolní otázky

4.7 K čemu lze využít kódování číslic kódem BCD?

4.8 Jaký je princip ukládání znakové informace v kódování UTF-8?

4.9 Jak se zjistí hodnota libovolného bitu v proměnné?

4.10 Jak se nastaví libovolný bit v proměnné na požadovanou hodnotu?

4.11 Jaký je princip implementace množiny čísel o rozsahu 0 až 32 (příp. 0 až 64)?

4.12 Jaká je délka exponentu u proměnných typu `float`? A jaká je u proměnných typu `double`?

4.13 Jakou bezpečnostní výhodu využíváme, zašifrujeme-li soubor operací XOR heslem zadaným pouze z klávesnice?