Práce s uživatelskými soubory

Programovací techniky

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

Standardní soubory – shrnutí

Standardní soubory – shrnutí

- **Ovládání**

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**
  - Data získána z klávesnice terminálu

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**
  - Data získána z klávesnice terminálu
  - Přesměrování vstupu/výstupu/chybového výstupu

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**
  - Data získána z klávesnice terminálu
  - Přesměrování vstupu/výstupu/chybového výstupu
  - Kolona

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**
  - Data získána z klávesnice terminálu
  - Přesměrování vstupu/výstupu/chybového výstupu
  - Kolona
- **Podpora v C++**

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**
  - Data získána z klávesnice terminálu
  - Přesměrování vstupu/výstupu/chybového výstupu
  - Kolona
- **Podpora v C++**
  - Knihovna iostream

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**
  - Data získána z klávesnice terminálu
  - Přesměrování vstupu/výstupu/chybového výstupu
  - Kolona
- **Podpora v C++**
  - Knihovna iostream
  - cin, cout, cerr, clog

Standardní soubory – shrnutí

- **Ovládání**
  - Automaticky otevřeny a uzavřeny, není problém s chybovými stavy
  - Jsou textové
  - Vstup a výstup je automaticky konvertován
  - Je možný i přenos bez konverze (znak po znaku)
- **Vazba na operační systém**
  - Data získána z klávesnice terminálu
  - Přesměrování vstupu/výstupu/chybového výstupu
  - Kolona
- **Podpora v C++**
  - Knihovna iostream
  - cin, cout, cerr, clog
  - getline, get

Uživatelské soubory

Uživatelské soubory

- Libovolný soubor na disku

Uživatelské soubory

- Libovolný soubor na disku
- Je potřebné zajistit spojení mezi proměnnou v programu a souborem na disku

Uživatelské soubory

- Libovolný soubor na disku
- Je potřebné zajistit spojení mezi proměnnou v programu a souborem na disku
- Je potřebné zajistit otevření a detekovat chybové stavy

Uživatelské soubory

- Libovolný soubor na disku
- Je potřebné zajistit spojení mezi proměnnou v programu a souborem na disku
- Je potřebné zajistit otevření a detekovat chybové stavy
- Je potřebné zajistit uzavření (zejména při zápisu)

Textové soubory

Textové soubory

- Prvkem je jeden znak, pracovat lze i s řádky

Textové soubory

- Prvkem je jeden znak, pracovat lze i s řádky
- Druhy znaků: zobrazitelné, řídicí, bílé

Textové soubory

- Prvkem je jeden znak, pracovat lze i s řádky
- Druhy znaků: zobrazitelné, řídicí, bílé
- Přenos dat s konverzemi, soubor čitelný člověkem

Textové soubory

- Prvkem je jeden znak, pracovat lze i s řádky
- Druhy znaků: zobrazitelné, řídicí, bílé
- Přenos dat s konverzemi, soubor čitelný člověkem
- **Manipulace**
  - Knihovna: `fstream`
  - Deklarace: `ifstream soubor; ofstream soubor;`
    `fstream soubor;`
  - Otevření: `soubor.open(”diskové jméno”);`
    lze spojit s deklarací:
    `fstream soubor (”diskové jméno”)`

Textové soubory

- **Manipulace – pokračování**
  - Čtení: uvažujme `string s; char c, p[N];`
    `soubor >> proměnná`
    `soubor.get(c); soubor.getline(p, bajtů, oddělovač);`
    `getline(soubor, s); soubor.read(p, bajtů)`
  - Zjištění stavu: `soubor.is_open()`, `soubor.eof()`
  - Zápis: `soubor << výraz`, zde lze použít `endl`
    `soubor.put(c); soubor.write(p, bajtů);`
  - Uzavření: `soubor.close();`

Příklad

- V jistém konfiguračním souboru s názvem „datarc“ je na každém řádku dvojice údajů: název parametru a jeho hodnota. Dvojice je oddělena rovnítkem. Přečtěte tento soubor a zjistěte, jakou hodnotu má parametr „MaxRowCount“.

Příklad

- V jistém konfiguračním souboru s názvem „datarc“ je na každém řádku dvojice údajů: název parametru a jeho hodnota. Dvojice je oddělena rovnítkem. Přečtěte tento soubor a zjistěte, jakou hodnotu má parametr „MaxRowCount“.
- Příklad datového souboru:

```
neco=78
param2=retezec
MaxRowCount=32
dalsipar=19
MaxRowCount=17
konec=10
```

Příklad

- Řešení:

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main(){
  ifstream konf (”datarc”);
  if (konf.is_open()) {
    //kontrola použitelnosti souboru
     char par[30]; //proměnné pro čtení
     string s;
     int num;
```

Příklad

- (pokračování řešení příkladu)

```
        while (not konf.eof()) {
          konf.getline(par, 30, '=');
          if (strcmp(par,"MaxRowCount")==0)
              konf>>num;
          else getline(konf, s);
        }
        cout << "Hodnota je " << num << endl;
    } else
        cerr << "Soubor datarc nelze otevřít."
             << endl;
    return 0;
}
```

Netextové (binární) soubory

- Jsou určeny pro libovolné typy dat; jsou nečitelné člověkem

Netextové (binární) soubory

- Jsou určeny pro libovolné typy dat; jsou nečitelné člověkem
- Výměna dat probíhá bez konverzí, tedy maximální rychlostí

Netextové (binární) soubory

- Jsou určeny pro libovolné typy dat; jsou nečitelné člověkem
- Výměna dat probíhá bez konverzí, tedy maximální rychlostí
- Otevření: soubor.open(”diskové jméno”, mód)

Netextové (binární) soubory

- Jsou určeny pro libovolné typy dat; jsou nečitelné člověkem
- Výměna dat probíhá bez konverzí, tedy maximální rychlostí
- Otevření: soubor.open(”diskové jméno”, mód)
- Mód: `ios::xxx`, kde XXX je:
  - `app`, nastaví pozici zápisu na konec;
  - `ate`, nastaví ukazatel do souboru na konec;
  - `binary`, soubor je binární;
  - `in`, soubor je otevřen pro čtení;
  - `out`, soubor je otevřen pro zápis;
  - `trunc`, obsah souboru je smazán a bude přepsán novými daty.

Netextové (binární) soubory

- Jsou určeny pro libovolné typy dat; jsou nečitelné člověkem
- Výměna dat probíhá bez konverzí, tedy maximální rychlostí
- Otevření: soubor.open(”diskové jméno”, mód)
- Mód: `ios::xxx`, kde XXX je:
  - `app`, nastaví pozici zápisu na konec;
  - `ate`, nastaví ukazatel do souboru na konec;
  - `binary`, soubor je binární;
  - `in`, soubor je otevřen pro čtení;
  - `out`, soubor je otevřen pro zápis;
  - `trunc`, obsah souboru je smazán a bude přepsán novými daty.
- Módy lze kombinovat operátorem „|“.

Netextové soubory

- Pro čtení a zápis předpokládejme tuto deklaraci:

```
fstream soubor; char pole[N];
```

Netextové soubory

- Pro čtení a zápis předpokládejme tuto deklaraci:

```
fstream soubor; char pole[N];
```

- **Čtení**

Netextové soubory

- Pro čtení a zápis předpokládejme tuto deklaraci:
  `fstream soubor; char pole[N];`
- **Čtení**
  - Pro čtení existuje metoda `soubor.read(pole, M)`

Netextové soubory

- Pro čtení a zápis předpokládejme tuto deklaraci:
  `fstream soubor; char pole[N];`
- **Čtení**
  - Pro čtení existuje metoda `soubor.read(pole, M)`
  - Lze však číst jakákoliv data, která je pouze potřebné přetypovat na typ `char*`. Druhý parametr je požadovaný počet přenášených bajtů. `soubor.gcount()` udává skutečný počet přečtených bajtů.

Netextové soubory

- Pro čtení a zápis předpokládejme tuto deklaraci:
  `fstream soubor; char pole[N];`
- **Čtení**
  - Pro čtení existuje metoda `soubor.read(pole, M)`
  - Lze však číst jakákoliv data, která je pouze potřebné přetypovat na typ `char*`. Druhý parametr je požadovaný počet přenášených bajtů. `soubor.gcount()` udává skutečný počet přečtených bajtů.
  - Příklad:

```
struct Data {
  int klic;
  double hodnota;
};
Data *x;
soubor.read((char*)x, sizeof(Data));
```

Netextové soubory

- **Zápis**

Netextové soubory

- **Zápis**
  - Podobně jako u čtení existuje metoda

    `soubor.write(pole, M)`

Netextové soubory

- **Zápis**
  - Podobně jako u čtení existuje metoda
    `soubor.write(pole, M)`
  - Příklad:

```
struct Data {
  int klic;
  double hodnota;
};
Data *x;
soubor.write((char*)&x, sizeof(Data));
```

Příklad

- Předpokládejme, že v paměti je naplněno pole záznamů o zboží. Chceme obsah pole uložit do binárního souboru s názvem „data.bin“.

Příklad

- Předpokládejme, že v paměti je naplněno pole záznamů o zboží. Chceme obsah pole uložit do binárního souboru s názvem „data.bin“.
- Definice datového typu pole záznamů:

```
const int MaxPole = 300;
struct TypZbozi {
  string Nazev;
  int PocKusu;
  float CenaKus;
};
typedef TypZbozi TypSklad[MaxPole];
TypSklad Sklad;
int Naplneno; //počet naplněných složek pole
```

Příklad

- Práce se souborem:

```
int *UkNaplneno=&Naplneno;
   //ukazatel vhodný k přetypování
fstream vystup;
vystup.open(”data”, ios::binary|ios::out);
   //binární soubor otevřený pro zápis
vystup.write((char*)UkNaplneno, sizeof(int));
  //totéž: vystup.write((char*)&Naplneno, ...
  //nejprve vypíšeme počet záznamů
vystup.write((char*)&Sklad,
   Naplneno*sizeof(TypZbozi));
  //a následně celé pole najednou
vystup.close();
  //soubor je nutné uzavřít
```

Ovládání pozice v souboru

- Čtení a zápis v souboru se typicky chovají jako **sekvenční** operace, tj. při každém vyvolání provedou přenos dat a automaticky posunou aktuální pozici.

Ovládání pozice v souboru

- Čtení a zápis v souboru se typicky chovají jako **sekvenční** operace, tj. při každém vyvolání provedou přenos dat a automaticky posunou aktuální pozici.
- Na binární soubor lze ovšem pohlížet i jako na pole bajtů, kde každý bajt má svůj index (číslováno od nuly).

Ovládání pozice v souboru

- Čtení a zápis v souboru se typicky chovají jako **sekvenční** operace, tj. při každém vyvolání provedou přenos dat a automaticky posunou aktuální pozici.
- Na binární soubor lze ovšem pohlížet i jako na pole bajtů, kde každý bajt má svůj index (číslováno od nuly).
- Otevřený soubor udržuje informaci o aktuální pozici čtení (bývá označována jako tzv. get pointer) a aktuální pozici zápisu (put pointer). Obě pozice jsou datového typu long int.

Ovládání pozice v souboru

- Čtení a zápis v souboru se typicky chovají jako **sekvenční** operace, tj. při každém vyvolání provedou přenos dat a automaticky posunou aktuální pozici.
- Na binární soubor lze ovšem pohlížet i jako na pole bajtů, kde každý bajt má svůj index (číslováno od nuly).
- Otevřený soubor udržuje informaci o aktuální pozici čtení (bývá označována jako tzv. get pointer) a aktuální pozici zápisu (put pointer). Obě pozice jsou datového typu long int.
- Obě pozice lze zjišťovat:

  `soubor.tellg()` a `soubor.tellp()`

Ovládání pozice v souboru

- Čtení a zápis v souboru se typicky chovají jako **sekvenční** operace, tj. při každém vyvolání provedou přenos dat a automaticky posunou aktuální pozici.
- Na binární soubor lze ovšem pohlížet i jako na pole bajtů, kde každý bajt má svůj index (číslováno od nuly).
- Otevřený soubor udržuje informaci o aktuální pozici čtení (bývá označována jako tzv. get pointer) a aktuální pozici zápisu (put pointer). Obě pozice jsou datového typu long int.
- Obě pozice lze zjišťovat:

  `soubor.tellg()` a `soubor.tellp()`
- Obě pozice lze také nastavit:

  `soubor.seekg(index)` a `soubor.seekp(index)`

Jednoduchý příklad

- Předpokládejme, že v paměti existuje naplněné pole celočíselných hodnot o VelPole prvcích. Toto pole vypíšeme do netextového souboru. Později se rozhodneme, že některý údaj v souboru potřebujeme za určitých podmínek aktualizovat – zadáme dvě hodnoty: první se má najít a nahradit v souboru tou druhou.

Jednoduchý příklad

- Předpokládejme, že v paměti existuje naplněné pole celočíselných hodnot o VelPole prvcích. Toto pole vypíšeme do netextového souboru. Později se rozhodneme, že některý údaj v souboru potřebujeme za určitých podmínek aktualizovat – zadáme dvě hodnoty: první se má najít a nahradit v souboru tou druhou.
- Řešení:

```
int main() {
  const int VelPole = 100;
  int pole[VelPole];
  for (int i=0; i<VelPole; i++) pole[i]=2*i;
     //pole je něčím naplněno...
  int Hledat, Nahradit, ZPole;
     //co hledáme, čím nahrazujeme, kam čteme
```

Pokračování příkladu

- Práce se souborem:

```
fstream bindata (”data.bin”, ios::binary
                 | ios::out);
   //soubor otevřeme pro zápis
bindata.write((char*)pole, VelPole*sizeof(int));
   //vypíšeme pole do souboru
bindata.close(); //soubor uzavřeme...
bindata.open(”data.bin”, ios::binary
                | ios::out | ios::in);
   //... a otevřeme pro čtení i zápis
cin >> Hledat >> Nahradit;
   //vstupní data
```

Pokračování příkladu

- Aktualizace dat v souboru:

```
  for (int i=0; i<VelPole; i++) {
    bindata.seekg(i*sizeof(int));
       //nastavíme pozici v souboru
    bindata.read((char*)&ZPole, sizeof(int));
       //přečteme, pozice se automaticky posune
    if (ZPole==Hledat) { //chceme aktualizovat
      bindata.seekp(i*sizeof(int)); //pozice
      bindata.write((char*)&Nahradit, sizeof(int))
        //přepíšeme hodnotu v souboru
    }
  }
  bindata.close(); //ukončení práce se souborem
  return 0;
}
```