Datové proudy – objektový vstup a výstup v C++

Petr Šaloun

10. listopadu 2014

Datové proudy – objektový vstup a výstup v C++

Základní pojmy

Znak je elementární textová informace, je umístěn v ASCII tabulce – číselný kód 8 bitový, datové typy znak char, řetězec string:

– tisknutelný znak 32 až 127,

– řídicí *escape sekvence*,

– znaky národních abeced.

16bitový kód UNICODE, datové typy znak wchar_t, řetězec wstring.

Slovo – posloupnost znaků. Slova jsou navzájem oddělena interpunkčními znaménky a odsazovači.

Řádek textu – posloupnost slov ukončená symbolem (symboly) přechodu na nový řádek.

Délka řádku?

Třída string vytvoří potřebný prostor.

Na úrovni OS možnost vstup a výstup přesměrovat – programy, které čtou ze standardního vstupu a zapisují do standardního výstupu, se nazývají *filtry*.

Textový I/O:

- Pro ukončení vstupu z klávesnice, Win32 *ctrl-z*, unix *ctrl-d*.

- Standardní objektový vstup a výstup používá vyrovnávací paměť jeden textový řádek. Program může vstup číst až v okamžiku, kdy ukončíme řádek klávesou *enter*.

- Hlavička `iostream`.

Třídy pro vstup a výstup

datový proud – *data stream*

textový, *binární*

hlavička **iostream**, třída **ios**

Třídy pro vstup a výstup jsou založeny na šablonách – stejné rozhraní pro vstup a výstup textu, bez ohledu na jeho kódování.

pomůcka: třídy a objekty vstup z proudu *i*, jako *input*; výstup *o*, od *output*; soubory *f*, jako *file*.

Typ **char**, šablony: **basic_** + název třídy:

třída	popis datového proudu
streambuf	s vyrovnávací pamětí
ios	vstupně výstupní
istream	vstupní
ostream	výstupní
fstream	souborový
ifstream	vstupní souborový
ofstream	výstupní souborový

Třídy pro standardní i souborové datové proudy.

Přetížené operátory pro vstup/výstup:

- **«** pro výstup do proudu, (*put to*, *insertion*),
- **»** pro vstup z proudu (*get from*, *extraction*).

Objekty standardního textového vstupu a výstupu

Standardní datové proudy C++ pro objektový I/O.

datový proud	popis	std. zařízení
cin	std. vstup	klávesnice
cout	std. výstup	obrazovka
cerr	std. chybový výstup	obrazovka
clog	std. chybový výstup po řádcích	obrazovka

Výstup a jeho formátování

operátor **«**

levý operand – objekt typu **ostream**,

pravý operand – libovolný typ (pro nějž je výstup definován).

```
cout << "Ahoj!" << endl;
int i = 5; double d = 123.456;
cout << "i =" << i << "\td=" << d << endl;
```

Formátovací příznaky pro textové proudy

příznak	popis
skipws	přeskoč bílé znaky na vstupu
left	výstup zarovnej vlevo
right	výstup zarovnej vpravo
internal	„vata“ po znaménku či indikátoru báze
dec	proveď desítkový převod
oct	proveď osmičkový převod
hex	proveď šestnáctkový převod
showbase	ukaž bázi při výstupu
showpoint	vynuť desetinnou tečku (rac. výstup)
uppercase	použij velká písmena při hex. výstupu
showpos	zobrazuj kladná celá čísla s ’+’
scientifi	použij např. 1.2345E2 pro rac. výstup
fixe	použij např. 123.45 pro rac. výstup
unitbuf	vyprázdni všechny vyr. paměti po každém přidán
stdio	po přidání vyprázdni stdout a stderr
boolalpha	umožní řetězec **true** či **false** při I/O
adjustfiel	společné pro internal, left right
basefiel	dec, oct, hex
floatfie	fixe a scientifi

Formátování výstupu pomocí stavových příznaků
formátovací stavové příznaky definov´ny ve třídě **ios** a v jejím *jmenném prostoru třídy*, proto např. **ios::hex**.

Výčtový datový typ pro konstanty všech formátovacích stavových příznaků **fmtflags** je definov´n ve třídě **ios**.

lze spojovat bitovým | nebo prostě sčítat – hodnoty jsou mocninami dvou.

metody **setf()** pro nastavení příznaků, či **unsetf()** pro jejich zrušení.

Objekt **cout** má přístup i k jiným metodám, než jen nastavení a zjištění stavu formátovacích příznaků. Jsou jimi zvláště metody **put()** a **write()**. I tyto metody umožňují výstup do textového proudu, který ovšem, na rozdíl od přetíženého operátoru **«**, není hodnotou formátovacích příznaků ovlivněn.

```
/**********************
 * soubor os-tflg.cpp
 **********************/

#include <iostream>

using namespace std;

void main() {
 cout << 123 << '\t' << 123.456e15 << '\t'
      << 0xfe << '\t' << true << endl;
 cout.setf(ios::hex | ios::showbase
                    | ios::boolalpha);
 cout.unsetf(ios::dec);
    // pro nektere prekladace nutne
 cout << 123 << '\t' << 123.456e15 << '\t'
      << 0xfe << '\t' << true << endl;
} // void main()
```

```
123     1.23456e+017    254     1
0x7b    1.23456e+017    0xfe    true
```

Formátování výstupu pomocí manipulátorů

manipulátory:

– speciální operátory podobné funkcím;

– používají jako svůj argument odkaz na proud, který rovněž vracejí;

– mohou být součástí příkazu výstupu.

Manipulátory textových proudů 1

manipulátor	význam	pro
boolalpha	log. hodnoty textově	I/O
dec	desítková soustava	I/O
endl	odřádkuje	O
ends	ukončí řet. a vyprázdní vyrov. paměť	O
fixe	nastaví příznak fixe	O
flus	vyprázdní vyr. paměť	O
hex	šestnáctková soustava	I/O
internal	nastaví příznak	O
left	desítková soustava	I/O
noboolalpha	zruší příznak	I/O
noshowbase	zruší příznak	O
noshowpoint	zruší příznak	O
noshowpos	zruší příznak	O
noskipws	zruší příznak	I
nounitbuf	zruší příznak	O
nouppercase	zruší příznak	O
oct	osmičková soustava	I/O

Manipulátory textových proudů 2

manipulátor	význam	pro
resetiosflags(fmtfla f)	zruší určené příznaky	I/O
right	nastaví příznak	O
scientifi	nastaví příznak	O
setbase(int base)	základ (0, 8, 10, 16)	I/O
setfill(in ch)	nastaví znak pro výplň	O
setiosflags(fmtfla f)	nastaví určené příznaky	I/O
setprecision(int p)	přesnost u rac. čísel	O
setw(int w)	šířka pole	O
showbase	nastaví příznak	O
showpoint	nastaví příznak	O
showpos	nastaví příznak	O
skipws	nastaví příznak	I
unitbuf	nastaví příznak	O
uppercase	nastaví příznak uppercase	O
ws	přeskočí uvozující bílé znaky	I

Pro vstup jsou definov´ny *další metody*: například **get()** a **read()**. Poslední načtený bajt je možno vrátit zpět pomocí metody **putback()**. Podívat se na příští vstupující znak bez jeho přečtení umožňuje metoda **peek()**. Poznamenejme ještě, že textové proudy mohou pro vstup i výstup využívat řádkovou vyrovnávací paměť – dokud neodřádkujeme, nezobrazí se z rozpracovaného výstupu do řádku nic.

```
// soubor stri-misc.cpp

#include <iostream>
#include <iomanip>

using namespace std;

int main() {
 bool b;
 cout << "zadej 'true' nebo 'false'" << endl;
 cin >> boolalpha >> b;
 cout << b << '\t' << boolalpha << b << endl;

 int d, o;
 cout << "zadej dve cela osmickova cisla"
      << endl;
 cin >> oct >> d >> o;

 char s[10];
 cout << "zadej retezec" << endl;
 cin >> setw(sizeof(s)) >> s;//s "nepretece"

 cout << endl << "d =" << d << "\to =" << o
      << endl << "s = " << s << endl;
} //int main()
```

```
zadej ’true’ nebo ’false’
false
0       false
zadej dve cela osmickova cisla
0123 0765
zadej retezec
Ahoj,babi!

d =83   o =501
s = Ahoj,babi
```

Textový vstup a výstup v paměti

sstream – hlavička pro práci s paměťovým textovým vstupem a výstupem,

istringstream, **ostringstream** – třídy pro textový paměťový vstup a výstup.

```
// soubor iostrm-mem.cpp
#include <iostream>
#include <sstream>
#include <string>

using namespace std;
int main() {
 string vstup = "22 7 Ahoj!",
                vystup, retezec;
 istringstream zpameti(vstup);
 int a, b;
 zpameti >> a >> b >> retezec;
 ostringstream dopameti(vystup);
 dopameti << a << " / " << b << " = "
          <<(double) a / b << endl;
 dopameti << retezec << endl;
 cout << dopameti.str();
} //int main()
```

```
22 / 7 = 3.14286
Ahoj!
```

Soubory

Soubor je posloupnost znaků (bajtů) ukončená nějakou speciální kombinací, která již k obsahu souboru nepatří – konec souboru, symbolicky **EOF**.

Textový soubor obsahuje *řádky textu*. *Binární soubor* obsahuje hodnoty ve stejném tvaru, v jakém jsou uloženy v paměti počítače.

vnější jméno souboru – jméno souboru na úrovni operačního systému;
vnitřní jméno souboru – identifikac souboru v rámci programu v jazyce C++, nejčastěji jde o jméno objektu, jehož prostřednictvím se souborem pracujeme.

Každý proud lze otevřít a uzavřít. Teprve po otevření můžeme s proudem pracovat. Při otevření proudu provádíme spojení mezi *vnitřním* a *vnějším jménem souboru*.

Při *otevření* určujeme režim našeho přístupu k datům v proudu.

Uzavřením proudu umožňujeme OS aktualizovat adresářové informace podle aktuálního stavu souboru, který byl s proudem spojen.

Objektový textový vstup a výstup z/do souborů

– shodný se standardním textovým vstupem a výstupem (do otevřených proudů):

```
double f;
vstupni_proud >> f;
vystupni_proud << "bylo nacteno :" << f;
```

– liší se hlavička:

```
#include <fstream>
```

a označení tříd

– **ofstream** pro výstup,

– **ifstream** pro vstup.

otevření souborového datového proudu– *konstruktor*:

```
ifstream    vstupni_proud("soubor.in");
ofstream   vystupni_proud("soubor.out");
```

nebo **open()**,
zavření **close()**.

neúspěšné otevření proudu – NULL:

```
if (vstupni_proud == 0)
  cerr  <<"chyba pri otevreni vstupniho "
          "souboru <soubor.in>";
else
  ...
```

Režimy práce se souborem (**ios::**):

režim	popis činnosti
app	připojuje data, vždy se zapisuje na konec souboru
ate	otevře a nastaví se na konec souboru
in	při otevření nastaví režim čtení (implicitní pro **ifstream**)
out	při otevření nastaví režim zápis (implicitní pro **ofstream**)
binary	otevře soubor v binárním režimu
trunc	zruší obsah souboru, pokud existuje

Při práci s binárním proudem je nezbytný *blokový přenos dat*.

náhodný přístup – položky binárního souboru mají známou velikost → lze vypočíst jejich polohu a přečíst, nebo zapsat na určenou pozici.

put(), **get()** pracují neformátovaně s jedním bajtem (znakem),
read(), **write()**: argumenty adresa a počet bajtů.

Metoda **eof()** vrací **true** při dosažení konce souboru.

```
// soubor strof-b1.cpp
// vytvori soubor odmoc.dta

#include <fstream>
#include <cmath>

using namespace std;

const char *jmeno = "odmoc.dta";

int main() {
 ofstream ofs(jmeno, ios::out | ios::binary);
 if (ofs != 0) {
   double f;
   for (int i = 0; i < 100; i++) {
     f = sqrt(i+1);
     ofs.write((const char *) &f, sizeof(f));
   } // for (;;)
   ofs.close();
 } // if (ofs != 0)
} //int main()
```

Sekvenční čtení z **odmoc.dta**

```
 +1.000 +1.414  +1.732  +2.000  +2.236  +2.449  +2.6
+3.317  +3.464  +3.606  +3.742  +3.873  +4.000  +4.1
+4.583  +4.690  +4.796  +4.899  +5.000  +5.099  +5.1
+5.568  +5.657  +5.745  +5.831  +5.916  +6.000  +6.0
+6.403  +6.481  +6.557  +6.633  +6.708  +6.782  +6.8
+7.141  +7.211  +7.280  +7.348  +7.416  +7.483  +7.5
+7.810  +7.874  +7.937  +8.000  +8.062  +8.124  +8.1
+8.426  +8.485  +8.544  +8.602  +8.660  +8.718  +8.7
+9.000  +9.055  +9.110  +9.165  +9.220  +9.274  +9.3
+9.539  +9.592  +9.644  +9.695  +9.747  +9.798  +9.8
```

```
// soubor strif-bs.cpp
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
const char *jmeno = "odmoc.dta";
int main() {
 ifstream ifs(jmeno, ios::in | ios::binary);
 if (ifs != 0) {
  double f;
  cout<<setw(7) << setprecision(4)
  <<setiosflags(ios::showpoint|ios::showpos);
  while (true) {
   if (ifs.eof()) { // test konce souboru
    break;
   } // if (ifs.eof())
  ifs.read((char *) &f, sizeof(f));
  cout << f << '\t';
  } // while (true)
  ifs.close();
 } // if (ifs != 0)
} //int main()
```

Souborový datový proud otevřený v binárním režimu můžeme číst *sekvenčně* i *náhodně*. Stejné je to se zápisem.

Pro kombinaci čtení a zápisu nad stejným souborem – dva ukazatele:
get ukazatel – ukazuje na místo v souboru, odkud proběhne příští čtení;
put ukazatel – na místo, na které proběhne příští zápis.

Metody rozhraní zpřístupňující popsaná ukazatele programátorovi se jmenují:
tellg(), **tellp()** vrací hodnoty ukazatelů,
seekg(), **seekp()** je nastavují.

pos_type – typ pro čtení pozice v souboru. Nastavení pozice: počet bajtů a vztažná pozice.

Konstanty pro určení vztažné pozice v souboru, **ios::**

identifik´tor	význam
beg	posun vůči aktuální pozici
cur	posun vzhledem počátku
end	posun vzhledem ke konci

```
ifs.seekg((i - 1) * sizeof(f), ios::beg);
```

```
// soubor strif-bn.cpp

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

const char *jmeno = "odmoc.dta";

int main() {
 ifstream ifs(jmeno, ios::in | ios::binary);
 if (ifs != 0) {
  double f;
  int i;
  cout<<setw(7) << setprecision(4)
   << setiosflags(ios::showpoint
       | ios::showpos);
  cout
   <<"Zadej cele cislo v rozsahu <1, 100>:";
  cin >> i;
```

```
  while ((i > 0) && (i < 101)) {
   ifs.seekg((i - 1) * sizeof(f), ios::beg);
   ifs.read((char *) &f, sizeof(f));
   cout << "druha odmocnina z " << i
        << "\tje " << f << endl;
   cout
    << "Zadej cele cislo v rozsahu <1, 100>:";
   cin >> i;
  } // while()
  cout << endl << "Konec." << endl;
  ifs.close();
 } // if (ifs != 0)
} //int main()
```

```
Zadej cele cislo v rozsahu <1, 100>:2
druha odmocnina z +2    je +1.414
Zadej cele cislo v rozsahu <1, 100>:4
druha odmocnina z +4    je +2.000
Zadej cele cislo v rozsahu <1, 100>:29
druha odmocnina z +29   je +5.385
Zadej cele cislo v rozsahu <1, 100>:0

Konec.
```