Потоци II

Любомир Чорбаджиев
Технологическо училище "Електронни системи"
Технически университет, София
lchorbadjiev@elsys-bg.org
Revision : 1.6

18 април 2005 г.

Форматиране

- Форматирането на входно/изходните операции се контролира чрез класовете `basic_ios` и `ios_base`.
- За управление на форматирането на входно/изходните операции се използва набор от флагове, определени в `ios_base`.
- Част от флаговете, определящи състоянието на формата са представени в следващият фрагмент:

Форматиране

```
class ios_base {
public:
 typedef implementation_dependent fmtflags;
 static const fmtflags
  skipws,     // пропуска разделителите при четене
  boolalpha, // типа boolean се представят
             // като true и false
  // целочислени типове
  dec,        // десетична система
  hex,        // шестнадесетична система
  oct,        // осмично система
  showbase,  // поставя префикс,
             // обозначаващ системата
  // числа с плаваща запетая
  scientific,// представяне във вида: d.dddddeddd
  fixed,      // представяне във вида: ddddd.dd
  showpoint, // незначеща нула пред десетичната точка
  showpos,   // явен знак '+' пред положителните числа
  ...;
};
```


Състояние на формата

`skipws`	пропускане на разделителните символи
`boolalpha`	представяне на типа boolean
`dec`	целочислени типове – в каква бройна
`hex`	система да се извеждат
`oct`	
`showbase`	представяне на бройната система
`scientific`	точка
`fixed`	как се извеждат типовете с плаваща
`showpoint`	
`showpos`	

Форматиране

- За манипулиране на състоянието на формата, в класа `ios_base` са дефинирани следните методи:

```
class ios_base {
public:
  ...
  fmtflags flags() const;
  fmtflags flags(fmtflags f);
  fmtflags setf(fmtflags f){
    return flags(flags()|f);
  }
  fmtflags setf(fmtflags f, fmtflags mask) {
    return flags((flags()&~mask)|(f&mask));
  }
  void unset(fmtflags mask) {
    flags(flags()&~mask);
  }
};
```


Форматиране

- Стандартната схема за работа с флаговете за форматиране е следната:
  - ◇ запомняме състоянието на формата;
  - ◇ променяме състоянието на формата и използваме потока;
  - ◇ възстановяваме предишното състояние на потока.

```
void foo(void) {
  ios_base::fmtflags old_flags=cout.flags();
  cout.setf(ios_base::oct);
  ...
  cout.flags(old_flags);
};
```


Извеждане на цели числа

- Добавянето на флагове чрез метода `setf()` или чрез побитово "ИЛИ" (|) е удобно, само когато дадена характеристика на потока се управлява от един бит.
- Тази схема е неудобна в случаи като определяне на бройната система. В такива случаи състоянието на формата не се определя от един бит.
- Решението на този проблем, което се използва в `<iostream>`, е да се предостави версия на `setf()` с втори "псевдо-аргумент":

```
cout.setf(ios_base::oct, ios_base::basefield);
cout.setf(ios_base::dec, ios_base::basefield);
cout.setf(ios_base::hex, ios_base::basefield);
```


Извеждане на цели числа: пример

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

int main(int argc,char* argv[]) {
  cout.setf(ios_base::showbase);
  cout.setf(ios_base::oct, ios_base::basefield);
  cout<< 1234 << '␣' << 1234 << endl;
  cout.setf(ios_base::dec, ios_base::basefield);
  cout<< 1234 << '␣' << 1234 << endl;
  cout.setf(ios_base::hex, ios_base::basefield);
  cout<< 1234 << '␣' << 1234 << endl;

  return 0;
};
```

```
02322 02322
1234 1234
0x4d2 0x4d2
```

Извеждане на числа с плаваща точка

- Извеждането на числа с плаваща запетая се определя от **формата** и **точността.**
- Форматите, които се използват за извеждане на числа с плаваща запетая са:
  - ◇ **Универсален** — позволява на потока сам да реши в какъв вид да се представи извежданото число. По подразбиране потоците използват този формат.
  - ◇ **Научен** — представя числото като десетична дроб с една цифра преди десетичната точка и показател на степента.
  - ◇ **Фиксиран** — точността определя максималният брой цифри след десетичната точка.
- По подразбиране точността е 6 цифри.

Извеждане на числа с плаваща точка: пример

```
1  #include <iostream>
2  using namespace std;
3  int main(int argc, char* argv[]) {
4    cout<<1234.56789<<'␣'<<1234.5678901<<endl;
5    cout.setf(ios_base::scientific,
6              ios_base::floatfield);
7    cout<<1234.56789<<'␣'<<1234.5678901<<endl;
8    cout.setf(ios_base::fixed,
9              ios_base::floatfield);
10   cout<<1234.56789<<'␣'<<1234.5678901<<endl;
11   cout.setf(static_cast<ios_base::fmtflags>(0),
12             ios_base::floatfield);
13   cout<<1234.56789<<'␣'<<1234.5678901<<endl;
14   return 0;
15 }
```

```
1234.57 1234.57
1.234568e+03 1.234568e+03
1234.567890 1234.567890
1234.57 1234.57
```


Извеждане на числа с плаваща точка

- За промяна на точността на работа с числа с плаваща запетая се използват следните методи:

```
class ios_base {
public:
  ...
  unsigned precision() const;
  unsigned precition(unsigned n);
  ...
};
```

- Използването на `precision()` влияе на всички входно/изходни операции с потока и действа до следващото използване на метода.

Извеждане на числа с плаваща точка: пример

```
1  #include <iostream>
2  using namespace std;
3  int main(int argc,char* argv[]) {
4    cout.precision(12);
5    cout<<123456789<<'␣'<<1234.12345<<'␣'
6        <<1234.123456789<<endl;
7    cout.precision(9);
8    cout<<123456789<<'␣'<<1234.12345<<'␣'
9        <<1234.123456789<<endl;
10   cout.precision(4);
11   cout<<123456789<<'␣'<<1234.12345
12       <<'␣'<<1234.123456789<<endl;
13   return 0;
14 }
```

```
123456789 1234.12345 1234.12345679
123456789 1234.12345 1234.12346
123456789 1234 1234
```

Полета за изход

```
#include <iostream>
using namespace std;
int main(int argc,char* argv[]) {
  cout << '(';
  cout.width(5);
  cout << 12 << ')' << endl;;
  cout << '(';
  cout.width(5); cout.fill('#');
  cout << 12 << ')' << endl;;
  cout << '(';
  cout.width(5); cout.fill('#');
  cout << "a" << ')' << endl;;
  cout << '(';
  cout.width(0); cout.fill('#');
  cout << "a" << ')' << endl;;
  return 0;
}
```

```
(   12)
(###12)
(####a)
(a)
```


Манипулатори

- Да се променя състоянието на потока посредством флаговете на формата е неудобно.
- Стандартната библиотека предоставя набор от функции и обекти за манипулиране на състоянието на потока – **манипулатори**.
- Основният начин за използване на манипулатори може да се види от следният пример:

  ```
  cout<<boolalpha<<true<<' '<<noboolalpha<<true;
  ```

  което извежда: `true 1`.
- Използват се и манипулатори с аргументи:

  ```
  cout<<setprecision(10)<<1234.12345678<<endl;
  ```

  което извежда: `1234.123457`. Манипулаторите с аргументи са дефинирани в `<iomanip>`.

Стандартни манипулатори

`boolalpha`	`noboolalpha`	представяне на типа boolean
`showbase`	`noshowbase`	представяне на бройната система
`showpoint`	`noshowpoint`	
`showpos`	`noshowpos`	
`skipws`	`noskipws`	пропускане на разделителните символи
`dec`		целочислени типове – в каква
`hex`		бройна система да се извеждат
`oct`		
`fixed`		как се извеждат типовете
`scientific`		с плаваща точка
`setprecision(n)`		точност на извеждането
`setw(int n)`		ширина на полето за извеждане
`setfill(int c)`		символ за запълване на полето

Файлови потоци

- Потоците за работа с файлове са дефинирани в `<fstream>`.
- Потокът за писане във файл е `basic_ofstream`.

```
template<class Ch, class Tr=char_traits<Ch> >
class basic_ofstream: public basic_ostream<Ch,Tr> {
public:
  explicit basic_ofstream(const char* p,
                          openmode m=out|trunc);
  bool is_open() const;
  void open(const char* p, openmode m=out|trunc);
  void close();
  ...
};
```

- Аналогични са дефинициите на другите два потока за работа с файлове — `basic_ifstream` и `basic_fstream`.

Файлови потоци

```
class ios_base {
public:
  typedef implementation_dependent1 openmode;
  static const openmode app,
    ate, binary, in, out, trunc;
  ...
};
```

`in`	отваряне за четене
`out`	отваряне за запис
`app`	запис на данните в края на файла; предизвиква отваряне на файла в режим `out`
`ate`	запис на данните в края на файла
`trunc`	унищожава предишното съдържание на файла

Файлови потоци

```
#include <iostream>
#include <fstream>
#include <cstdlib>
void error(const char* p, const char* p2="") {
  std::cerr << p << ' ' << p2 << std::endl;
  std::exit(1);
}
int main(int argc,char* argv[]) {
  if(argc!=3) error("bad number of arguments...");

  std::ifstream from(argv[1]);
  if(!from) error("bad input file", argv[1]);
  std::ofstream to(argv[2]);
  if(!to) error("bad output file:", argv[2]);

  char ch;
  while(from.get(ch)) to.put(ch);

  if(!from.eof() || !to)
    error("something strange...");
  return 0;
}
```