

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:

```

1 class ios_base {
2 public:
3     typedef implementation_dependent fmtflags;
4     static const fmtflags
5         skipws,      // пропуска разделителите при четене
6         boolalpha,  // типа boolean се представят
7                     // като true и false
8     // целочислени типове
9     dec,            // десетична система
10    hex,            // шестнадесетична система
11    oct,            // осмично система
12    showbase,      // поставя префикс,
13                  // обозначаващ системата
14    // числа с плаваща запетая
15    scientific,    // представяне във вида: d.dddddeddd
16    fixed,         // представяне във вида: ddddd.dd
17    showpoint,     // незначеща нула пред десетичната точка
18    showpos,      // явен знак '+' пред положителните числа
19    ...;
20 };

```

2

Състояние на формата

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

3

Форматиране

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

```
1 class ios_base {
2 public:
3     ...
4     fmtflags flags() const;
5     fmtflags flags(fmtflags f);
6     fmtflags setf(fmtflags f){
7         return flags(flags()|f);
8     }
9     fmtflags setf(fmtflags f, fmtflags mask) {
10        return flags((flags()&~mask)|(f&mask));
11    }
12    void unset(fmtflags mask) {
13        flags(flags()&~mask);
14    }
15 };
```

4

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.

```
1 void foo(void) {
2     ios_base::fmtflags old_flags=cout.flags();
3     cout.setf(ios_base::oct);
4     ...
5     cout.flags(old_flags);
6 };
```

5

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);
```

6

Извеждане на цели числа: пример

```
1 #include <iostream>  
2 using namespace std;  
3  
4 int main(int argc, char* argv[]) {  
5     cout.setf(ios_base::showbase);  
6     cout.setf(ios_base::oct, ios_base::basefield);  
7     cout << 1234 << ' ' << 1234 << endl;  
8     cout.setf(ios_base::dec, ios_base::basefield);  
9     cout << 1234 << ' ' << 1234 << endl;  
10    cout.setf(ios_base::hex, ios_base::basefield);  
11    cout << 1234 << ' ' << 1234 << endl;  
12  
13    return 0;  
14 };
```

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

7

- 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 точка⁸: 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.

10

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
```

11

```

1 #include <iostream>
2 using namespace std;
3 int main(int argc, char* argv[]) {
4     cout << '(';
5     cout.width(5);
6     cout << 12 << ') ' << endl;;
7     cout << '(';
8     cout.width(5); cout.fill('#');
9     cout << 12 << ') ' << endl;;
10    cout << '(';
11    cout.width(5); cout.fill('#');
12    cout << "a" << ') ' << endl;;
13    cout << '(';
14    cout.width(0); cout.fill('#');
15    cout << "a" << ') ' << endl;;
16    return 0;
17 }

```

```

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

```

12

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>.

13

Стандартни манипулатори

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

14

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`.

15

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

16

```
1 #include <iostream>  
2 #include <fstream>  
3 #include <cstdlib>  
4 void error(const char* p, const char* p2="") {  
5     std::cerr << p << ' ' << p2 << std::endl;  
6     std::exit(1);  
7 }  
8 int main(int argc, char* argv[]) {  
9     if(argc!=3) error("bad number of arguments...");  
10  
11     std::ifstream from(argv[1]);  
12     if(!from) error("bad input file", argv[1]);  
13     std::ofstream to(argv[2]);  
14     if(!to) error("bad output file:", argv[2]);  
15  
16     char ch;  
17     while(from.get(ch)) to.put(ch);  
18  
19     if(!from.eof() || !to)  
20         error("something strange...");  
21     return 0;  
22 }
```

17