

Форматиране на вход и изход, Типове

Изготвил:
гл.ас. д-р Нора Ангелова

Форматиране на вход и изход

Форматиране на входа и изхода

Осъществява се чрез подходящи манипулатори.

```
cin >> манипулатор;
```

```
cin >> манипулатор(параметри);
```

```
cout << манипулатор;
```

```
cout << манипулатор(параметри);
```

```
#include <iomanip>
```

Форматиране на вход и изход

Манипулатор за задаване на ширина на полето на следващия вход/изход

Синтаксис

setw(<цял_израз>)

Семантика

Стойността на <цял_израз> задава ширината на полето на следващия вход/изход.

Форматиране на вход и изход

Пример: Нека имаме дефиницията
`int i = 1234, j = 9876;`

Изразът

```
cout << setw(10) << i << setw(10) << j << "\n";
```

Извежда

```
*****1234*****9876
```

където със * е означен символът интервал.

Форматиране на вход и изход

Манипулатори на позиционна система (dec, oct и hex)

dec – интерпретира цяло число в 10 п.с.

oct – интерпретира цяло число в 8 п.с.

hex - интерпретира цяло число в 16 п.с.

Използват се както при изход, така и при вход на целочислени данни/променливи.

Форматиране на вход и изход

Манипулатори на позиционна система (dec, oct и hex)

Пример 1.

```
int i = 12, j = 23;  
cout << setw(10) << dec << i << setw(10) << j << "\n";  
cout << setw(10) << oct << i << setw(10) << j << "\n";  
cout << setw(10) << hex << i << setw(10) << j << "\n";
```

Резултат:

```
*****12*****23  
*****14*****27  
*****c*****17
```

Форматиране на вход и изход

Манипулатори dec, oct и hex

Пример 2.

```
int i, j;  
cin >> hex >> i >> j;  
cout << setw(10) << dec << i << setw(10) << j << "\n";  
cout << setw(10) << oct << i << setw(10) << j << "\n";  
cout << setw(10) << hex << i << setw(10) << j << "\n";
```

Вход: c 17

Резултат:

```
*****12*****23  
*****14*****27  
*****c*****17
```

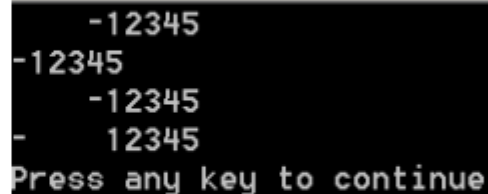
Форматиране на вход и изход

Други манипулатори:

а) За задаване на режим за изравняване (само при изход) – left, right (подразбира се) и internal;

Пример:

```
int x = -12345;  
cout << setw(10) << x << endl;  
cout << left << setw(10) << x << endl;  
cout << right << setw(10) << x << endl;  
cout << internal << setw(10) << x << endl;
```



```
-12345  
-12345  
-12345  
- 12345  
Press any key to continue
```


Форматиране на вход и изход

б) За задаване на символ за запълване на полето (само при изход): `setfill(char = ' ');`

Пример:

```
int x = 12345;
```

```
cout << setw(10) << setfill('.') << x << endl;
```

.....12345

Проблеми при работа с цели числа

Проблеми при работа с цели числа

Ако при работа с цели числа се получи стойност, която е извън размера на отделената памет, резултатът се препълва.

Не се съобщава за грешка.

Вместо това резултатът се отрязва, за да се побере в паметта за типа `int`, давайки стойност, която най-вероятно не е това, което се очаква.

Проблеми при работа с цели числа

Пример:

Известно е, че сумата на две положителни цели числа е положително цяло число.

В програмирането това не винаги е така.

Проблеми при работа с цели числа

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10000000000, y = 20000000000;
    cout << x << " and " << y << " are positive \n";
    cout << "The sum " << (x + y) << " is positive. \n";
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
10000000000 and 20000000000 are positive
The sum -1294967296 is positive.
Press any key to continue . . .
```

// с използване на assert

#include <iostream>

#include <cassert>

using namespace std;

int main()

{ **int** x = 1000000000, y = 2000000000;

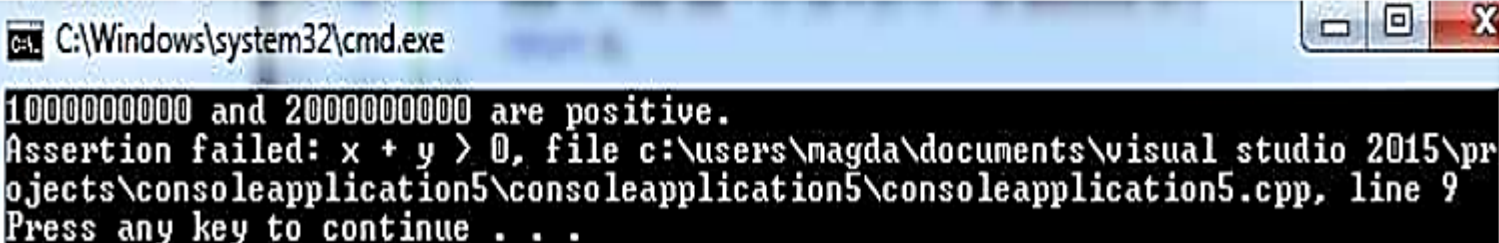
assert(x > 0 && y > 0);

cout << x << " and " << y << " are positive.\n";

assert (x + y > 0);

cout << "The sum " << (x + y) << " is positive.\n";

return 0;

} 

**Причина за проблема:
ПРЕПЪЛВАНЕ**

**Разрешаване на проблема:
Използване на „по-голям“ тип.**

Проблеми при работа с цели числа

Други цели типове

Други цели типове се получават от *int* като се използват модификаторите *short*, *long*, *signed* и *unsigned*.

Тези модификатори доопределят някои аспекти на типа *int*.

Проблеми при работа с цели числа

Други цели типове

short int	-32768 до 32767	2 байта
unsigned short int	0 до 65535	2 байта
long int	-2147483648 до 2147483647	4 байта
unsigned long int	0 до 4294967295	4 байта
unsigned int	0 до 4294967295	4 байта
long long int	-9,223,372,036,854,775,808 до 9,223,372,036,854,775,807	8 байта

Проблеми при работа с цели числа

Други целочислени типове

Type Name	Bytes	Other Names	Range of Values
int	4	signed	-2,147,483,648 to 2,147,483,647
unsigned int	4	unsigned	0 to 4,294,967,295
__int8	1	char	-128 to 127
unsigned __int8	1	unsigned char	0 to 255
__int16	2	short, short int, signed short int	-32,768 to 32,767
unsigned __int16	2	unsigned short, unsigned short int	0 to 65,535
__int32	4	signed, signed int, int	-2,147,483,648 to 2,147,483,647
unsigned __int32	4	unsigned, unsigned int	0 to 4,294,967,295
__int64	8	long long, signed long long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned __int64	8	unsigned long long	0 to 18,446,744,073,709,551,615

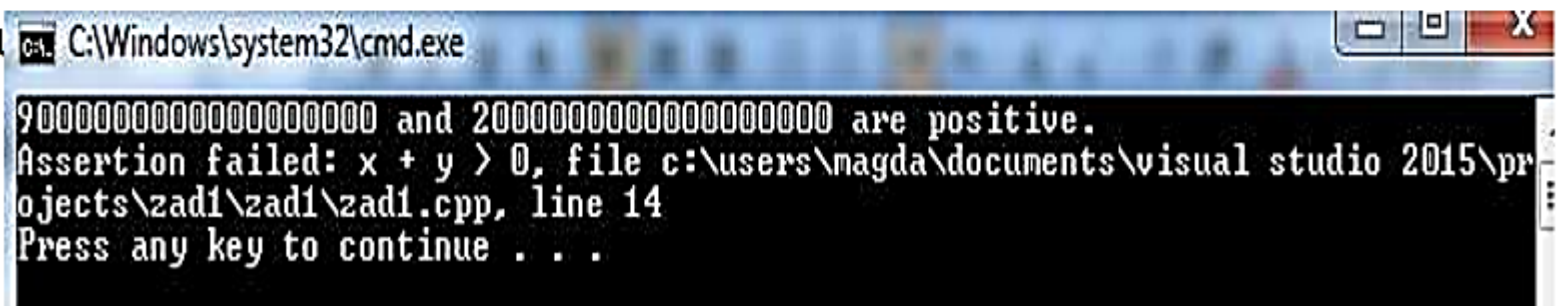
Проблеми при работа с цели числа

Опит за поправка на грешката

```
#include <iostream>
#include <cassert>
using namespace std;
int main()
{
    long long x = 10000000000, y = 20000000000;
    assert(x > 0 && y > 0);
    cout << x << " and " << y << " are positive.\n";
    assert(x + y > 0);
    cout << "The sum " << (x + y) << " is positive.\n";
    return 0;
}
```

Проблеми при работа с цели числа

```
#include <iostream>
#include <cassert>
using namespace std;
int main()
{ long long x = 9000000000000000000, y = 2000000000000000000;
  assert(x > 0 && y > 0);
  cout << x << " and " << y << " are positive.\n";
  assert(x + y > 0);
  cout << "The sum " << (x + y) << " is positive.\n";
  ret
```



```
C:\Windows\system32\cmd.exe
9000000000000000000 and 2000000000000000000 are positive.
Assertion failed: x + y > 0, file c:\users\magda\documents\visual studio 2015\pr
jects\zad1\zad1\zad1.cpp, line 14
Press any key to continue . . .
```

Проблеми при работа с цели числа

Проблемите произтичат от избраното:

- двоично представяне и
- ограничения брой битове за представяне.

**Ако по-голям целочислен тип не съществува,
може да се използва типът `double`.**

Реални типове

Тип double

$\langle \text{реално_число} \rangle ::=$

$\langle \text{цяло_число} \rangle . \langle \text{цяло_число_без_знак} \rangle |$

$\langle \text{цяло_число} \rangle \mathbf{E} \langle \text{порядък} \rangle |$

$\langle \text{цяло_число} \rangle . \langle \text{цяло_число_без_знак} \rangle \mathbf{E} \langle \text{порядък} \rangle$

$\langle \text{порядък} \rangle ::= \langle \text{цяло_число} \rangle$

Диапазон: $-1,74 \cdot 10^{308}$ до $1,7 \cdot 10^{308}$; има около 15 значещи десетични цифри.

Примери: 123e-2; -3054.59; 9.34E12; 4023.68087

Тип double

Представяне в паметта:

- нормализирано представяне във формат с плаваща запетая, т.е.

$$m.p^n$$

където

- m се нарича мантиса и удовлетворява $1 \leq m < p$;
- p е основата на бройната система, $p = 2$;
- n е цяло число, означаващо порядъка.

В различните типове компютри се използват различни варианти на това представяне, но най-често този формат се придържа към стандарта IEEE 754.

Tun double

Допълнение:

Сравнението за равенство на две реални числа x и y се реализира обикновено чрез проверка на неравенството:

$|x - y| < \epsilon$, където $\epsilon = 10^{-14}$ за типа *double*.

Използва се и релацията:

$$\frac{|x - y|}{\max\{|x|, |y|\}} \leq \epsilon$$

При аргумент от тип *double*, следващите функции връщат резултат от тип *double*.

Задаване на точност на формата на реалните числа

Точност в:

- основен (общ) формат;
- експоненциален (научен, *scientific*) формат;
- фиксиран (с фиксирана точка, *fixed*)

Пример. Реалното число 38.59417862 с точност 7 се представя по следния начин:

- в основен формат – броят на цифрите в цялата и дробната част е 7
38.59418
- в експоненциален формат (*scientific*) – броят на цифрите след десетичната точка на мантисата е 7
3.8594179e+001
- във фиксиран формат (*fixed*) – броят на цифрите след десетичната точка е 7
38.5941786

Задаване на точност на формата на реалните числа

Преминаване от един формат в друг:

Манипулатори

- scientific – за преминаване в експоненциален формат;*
- fixed – за преминаване във фиксиран формат;*
- setiosflags(ios::scientific) – за преминаване в експоненциален формат;*
- setiosflags(ios::fixed) – за преминаване във фиксиран формат;*
- resetiosflags(ios::scientific) – преминава се в основен формат;*
- resetiosflags(ios::fixed) – преминава се в основен формат.*

Задаване на точност

Задаване на точност

setprecision(int)

– аргументът е цял израз и задава точността в десетична позиционна система.

Семантика

Задава точност на реалните числа. Тази точност е в сила до указване на следваща. Възможно е закръгляване.

Забележка. Точността, която се подразбира е 6.

Задаване на точност - пример

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{ double x = 38.59417862;
  cout << setprecision(7);
  cout << x << endl;
  cout << scientific << x << endl;
  cout << fixed << x << endl;
  return 0;
}
```

Резултат:

38.59418

3.8594179e+001

38.5941786

Задаване на точност - пример

`double root2 = sqrt(2);`

Представяне на `root2` с точност 0, 1, 2, ..., 9

в основен формат:

1.41421
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356

в научен формат:

1.414214e+000
1.4e+000
1.41e+000
1.414e+000
1.4142e+000
1.41421e+000
1.414214e+000
1.4142136e+000
1.41421356e+000
1.414213562e+000

<i>Манипулатор</i>	<i>Предназначение</i>	<i>Използва се при</i>
boolalpha	Въвежда/извежда булевите константи като низове true или false.	вход и изход
showbase	Показва базата на числовата система с префикс 0, ако числото е осмично и с префикс 0x, ако числото е шестнадесетично.	изход
showpoint	Показва десетичната точка в записа на реалните числа.	изход
showpos	Показва знака + пред положително число.	изход
skipws	Пропуска незначещите символи (интервали, табулации, преминаване на нов ред) при въвеждане.	вход
unitbuf	Изчиства буфера след вмъкване (извеждане).	изход
uppercase	Предизвиква да се използват главни букви при шестнадесетичното представяне на цяло число или на главно Е в научния формат на реално число.	изход

Други манипулатори - пример

```
#include <iostream>
using namespace std;
int main()
{ int n = 19;
  cout << hex << showbase << n << endl;
  cout << noshowbase << n << endl;
  return 0;
}
```

Резултатът:

0x13

13

Други манипулатори - пример

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{ double a = 25.0, b = 137.0, pi = 3.1416;
  cout << setprecision (5);
  cout << showpoint << a << ' ' << b << ' ' << pi << endl;
  cout << noshowpoint << a << ' ' << b << ' '
      << pi << endl;
  return 0;
}
```

Резултат:

25.000 137.00 3.1416

25 137 3.1416

Точността (в смисъл правилността, коректността) на типа *double* е около 15 значещи цифри. Това може да се види от примера по-долу.

Пример.

```
#include <iostream>
using namespace std;
int main()
{ double a = 5e14;
  double b = a - 0.1;
  double c = a - b;
  cout << c << "\n";
  return 0;
}
```

Стойността на променливата *c* трябва да е 0.1, а програмата намира за такава 0.125. Причината е в броя на значещите цифри на *b*.

Пример.

```
#include <iostream>
using namespace std;
int main()
{ double a = 5e13;
  double b = a - 0.1;
  double c = a - b;
  cout << c << "\n";
  return 0;
}
```

Стойността на променливата *c* трябва да е 0.1, а програмата намира за такава 0.101563.

Пример.

```
#include <iostream>
using namespace std;
int main()
{ double a = 5e12;
  double b = a - 0.1;
  double c = a - b;
  cout << c << "\n";
  return 0;
}
```

Стойността на променливата *c* трябва да е 0.1, а програмата намира за такава 0.0996094. За *a* до 5e9 стойността на *c* е 0.1.