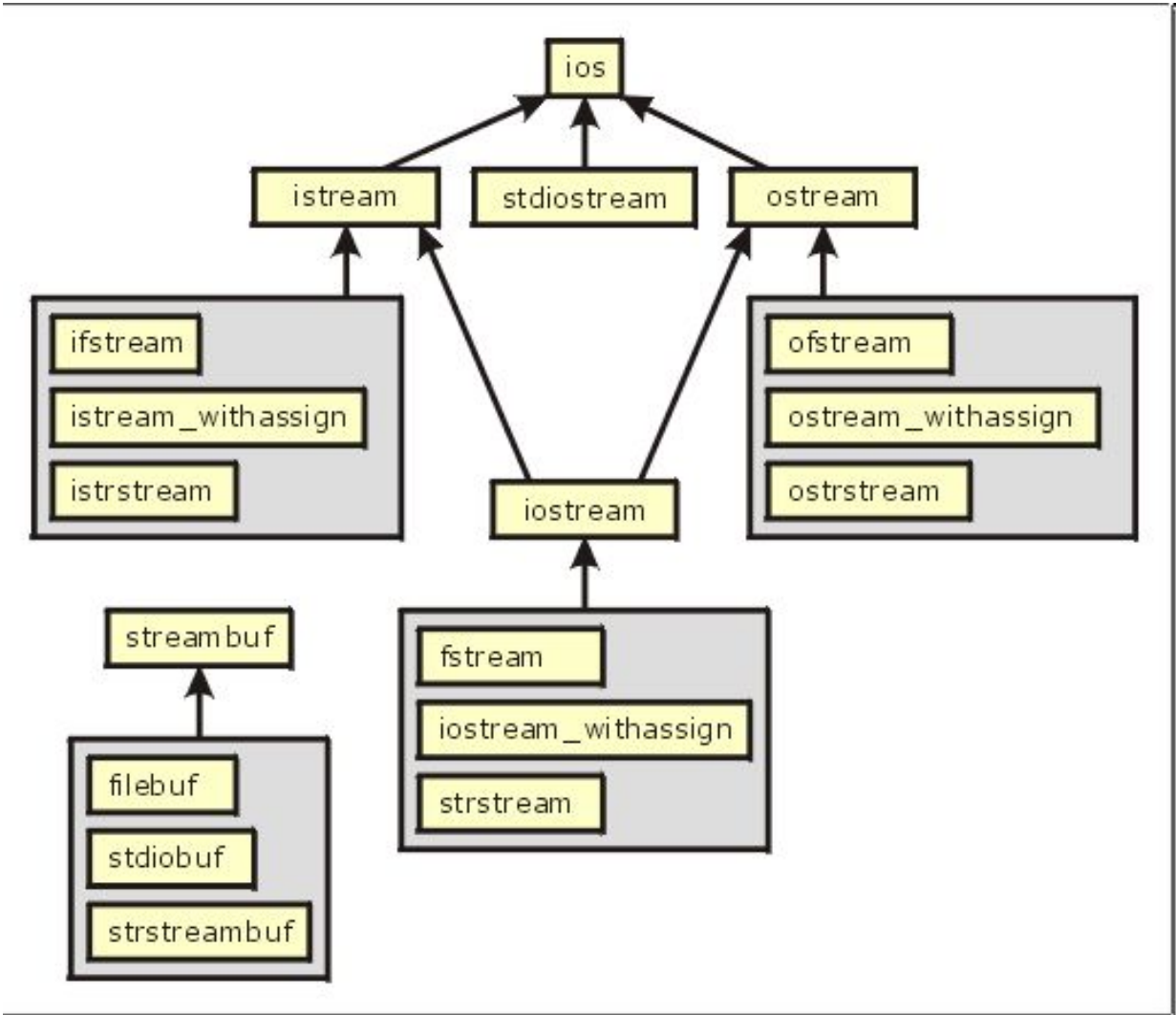


C++

Ievade un izvade

Internet adresses

- <http://www.cppreference.com/wiki/start>
- <http://www.cplusplus.com/doc/tutorial/>



Standarta ievade un izvade

1. `#include <iostream>`

2. Elementi

- Objekti `cin`, `out`
- Operatori `>>` un `<<`.

```
cout << "Five = " << 5;
```

Formatēta izvade

- Izdrukājamās informācijas formatēšanu veic:
 - izmantojot manipulatorus (izdrukas priekšraksta ietvaros);
 - izmantojot objekta cout **funkcijas** (pirms izdrukas priekšraksta).

iomanip.h

- Manipulatoriem vajadzīga bibliotēka iomanip.h.
- Formatēšanas funkcijām - nē!

Formatēšanas komandu darbības apgabals

- Lielākā daļa formatēšanas komandu darbojas no uzstādīšanas brīža līdz atsaukšanai vai programmas beigām (*piemēram, precision*),
- Daļa komandu attiecas tikai uz nākošo izdrukas operatoru << (pat ne uz visu nākošo izdrukas priekšrakstu, *piemēram, width*).

width/setw

- Funkcija/manipulators
- Uzstāda izvades apgabala platumu simbolos.
- **Darbojas tikai uz nākošo izvadāmo vērtību.**
- Pēc noklusēšanas izslēgts, t.i., izmanto tik daudz vietas, cik nepieciešams.

Piemērs

- Nākošajā piemērā 'B' tiek izdrukāts uzreiz pēc 'A'.
- 'C' tiek izdrukāts 3-vietīgā apgabalā.
- 'D' 12-vietīgā apgabalā.
- Pēc noklusēšanas apgabala ietvaros **pielīdzinot pie labās malas.**

```
cout << "A" << "B";  
cout << setw(3) << "C";  
cout.width(12);  
cout << "D" << endl;
```

```
AB  C  D
```

fill/setfill

- Uzstāda aizpildīšanas simbolu.
- Ir nozīme gadījumā, ja ir uzstādīts arī platums.
- Pēc noklusēšanas - tukšums.

Piemērs

```
cout.fill('_');  
cout << setw(10) << 123;  
cout << setfill('^') << setw(10) << 456 << endl;
```

```
_____123^^^^^^456
```

precision/setprecision

- Funkcija/manipulators
- Uzstāda skaitļa precizitāti vai nu ciparos aiz komata, vai nu zīmīgajos ciparos.
- Pēc noklusēšanas 6 zīmīgie cipari.

Piemērs

- Sekojošais piemērs parāda skaitļa izdruku 3 variantos:
 - ar 6 **zīmīgajiem** cipariem (pēc noklusēšanas),
 - ar 5 (ar manipulatora uzstādījumu) un
 - ar 4 (ar funkciju).

```
cout << 271.84753 << endl;  
cout << setprecision(5) << 271.84753 << endl;  
cout.precision(4);  
cout << 271.84753 << endl;
```

```
271.848  
271.85  
271.8
```

Formatēto stāvokļu karogi

- The `ios` class defines an enumeration of format state flags that you can use to affect the formatting of data in USL I/O streams. The following list shows the formatting features and the format flags that control them:
- Whitespace and padding: `ios::skipws`, `ios::left`, `ios::right`, `ios::internal`
- Base conversion: `ios::dec`, `ios::hex`, `ios::oct`, `ios::showbase`
- Integral formatting: `ios::showpos`
- Floating-point formatting: `ios::fixed`, `ios::scientific`, `ios::showpoint`
- Uppercase and lowercase: `ios::uppercase`
- Buffer flushing: `ios::stdio`, `ios::unitbuf`

Peldošā punkta formatēšana

`setf(ios::fixed)/fixed`

- Uzstādītās precizitātes noteikšana.
- Cik ciparu aiz komata izvadīt (nevis zīmīgos ciparus).

Piemērs

```
cout.precision(2);  
cout.setf(ios::fixed);  
cout << 271.84753 << endl;
```

```
cout.precision(3);  
cout << fixed << 271.84753 << endl;
```

```
271.85  
271.848
```

setf(ios::scientific)/scientific

- Atšķirībā no citu pāru parauga, nedara precīzi vienu un to pašu.
 - *scientific* - manipulators, kas nosaka zinātnisko skaitļa izvades formātu.
 - *setf(ios::scientific)* - funkcija, kas nosaka, ka uzstādītā precizitāte attiecas uz zīmīgajiem cipariem (nevis cipariem aiz komata).

Piemērs

- Komandas `precision(3)` un `setf(ios::fixed)` uzstāda precizitāti - 3 zīmes aiz komata.
- Komanda `setf(ios::scientific)` šo 3 ciparu precizitāti pārliedz uz zīmīgajiem cipariem.
- `scientific` nosaka zinātnisko pierakstu un tā ietvaros 3 ciparus aiz komata.

```
cout.precision(3);  
cout.setf(ios::fixed);  
cout << 271.84753 << endl;  
cout.setf(ios::scientific);  
cout << 271.84753 << endl;  
cout << scientific << 271.84753 << endl;
```

```
271.848  
272  
2.718e+002
```

Piemērs

- `cout.setf(ios::scientific,ios::floatfield);`
- `cout.setf(ios::fixed,ios::floatfield);`
- `cout.setf(0,ios::floatfield); // sākuma režīms`

```
cout << 1234.56789 << "\n";  
cout.setf(ios::scientific,ios::floatfield);  
cout << 1234.56789 << "\n";  
cout.setf(ios::fixed,ios::floatfield);  
cout << 1234.56789 << "\n";
```

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

Piemēri cout.setf(ios::showpoint)

```
float x = 18.0;
cout<< x << endl;           //displays 18
cout.setf(ios::showpoint);
cout<< x << endl;           //displays 18.0000
cout.setf(ios::scientific);
cout<< x << endl;           //displays
1.800000e+001
cout.unsetf(ios::showpoint);
cout.unsetf(ios::scientific);
cout<<x<<endl;             //displays 18
```

```
cout.setf(ios::fixed);       //Sets the cout flag for fixed-point
                             //non-scientific notation trailing zeros
cout.setf(ios::showpoint);  //Always shows decimal point
cout << setprecision(2);     //Two decimal places
cout << 5.8;
```

left, right

- Manipulatori pielīdzināšanai pie izdrukas apgabala kreisās vai labās malas.
- Ir nozīme tikai gadījumā, ja ir uzstādīts platums ar *width/setw*.
- `cout.setf(ios::left, ios::adjustfield); // left`
- `cout.setf(ios::right, ios::adjustfield); // right`
- `cout.setf(ios::internal, ios::adjustfield); // iekšējā`

Piemērs

```
cout.fill('*');  
cout << left << setw(10) << 123 << endl;  
cout << right << setw(10) << 456 << endl;
```

```
123*****  
*****456
```

Piemērs

```
cout.width(4);  
cout << '(' << -12 << ")\n"; cout.width(4);  
cout.setf(ios::left,ios::adjustfield);  
cout << '(' << -12 << ")\n";  
cout.width(4);  
cout.setf(ios::internal,ios::adjustfield);  
cout << '(' << -12 << ")\n";
```

```
(-12)  
( -12 )  
(-12)
```


Bāzes konvertācija

```
cout.setf(ios::oct,ios::basefield); // 8
```

```
cout.setf(ios::dec,ios::basefield); //10
```

```
cout.setf(ios::hex,ios::basefield); // 16
```

```
cout.setf(ios::showbase); // pirms visiem ieliek rāda katram
```

```
cout << 1234 << ' '; // 10
```

```
cout << 1234 << ' ';
```

```
cout.setf(ios::oct,ios::basefield); // 8
```

```
cout << 1234 << ' '; cout << 1234 << ' ';
```

```
cout.setf(ios::hex,ios::basefield); // 16
```

```
cout << 1234 << ' '; cout << 1234 << ' ';
```

```
1234 1234 2322 2322 4d2 4d2
```

Piemērs

```
int a=9;
    cout.setf(ios::oct,ios::basefield);
    cout << a << endl;
//vai
cout << oct << a << endl;
```

```
11
11
```

Formatētais ievads

```
cin >> i;
```

```
cin >> k;
```

```
cin >> i >> k;
```

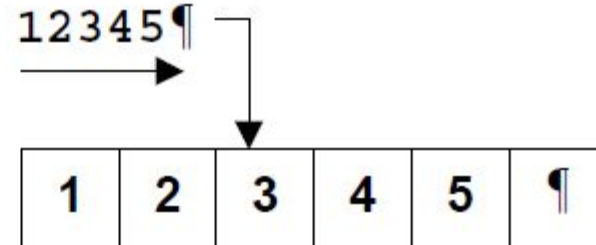
Ievades process

- Ievades process notiek 2 posmos:
 - informācijas nolasīšana no klaviatūras un uzkrāšana buferī līdz pat ENTER nospiešanai,
 - bufera satura pārstaigāšana, mēģinot izvilkt noteikta tipa vērtību vai vērtības, lai ierakstītu mainīgajā vai mainīgajos.

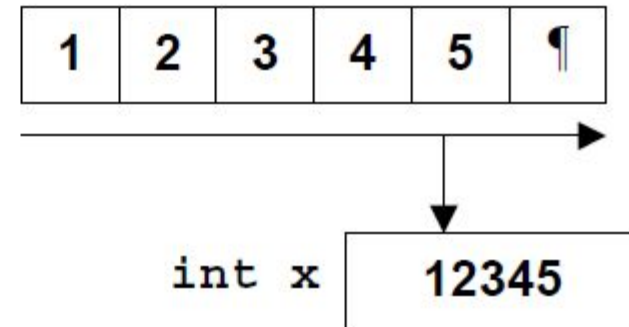
Vienkāršota ievades divdaļīgā darbības shēma

```
int x;  
cin >> x;
```

1. solis.
Pēc ENTER nospiešanas
informācija tiek ierakstīta
buferī.



2. solis.
No bufera tiek mēģināts
nolasīt noteikta tipa vērtību
un ierakstīt mainīgajā.



Ievades kļūdu apstrāde

- Pēc katras lasīšanas operācijas ir iespējams pārbaudīt vai tā bijusi **veiksmīga** vai **nē**.
- To veic ar objekta *cin* funkciju *good*.
- Iespējams pārbaudīt arī uz neveiksmi ar funkcijām *bad* vai *fail*.

```
int i;  
cin >> i;  
if (cin.good()) cout << "OK: " << i << endl;  
else cout << "ERROR" << endl;
```

```
123  
OK: 123  
abc  
ERROR
```

Piemērs

- Ja nolasīšana bijusi neveiksmīga, objekts `cin` tiek nobloķēts.
- Ja to neatbloķē, visas turpmākās darbības ar to tiek ignorētas.
- Piemērā redzams, ka pēc kļūdas komandā `cin >> i` (mēģinot ievadīt tekstu "abc" skaitliskā tipa mainīgajā), nākošā ievades komanda (`cin >> k`) tiek ignorēta (resp., otrreiz ievade no klaviatūras netiek prasīta).
- Abas izdrukas komandas nostrādā normāli, izdrukājot mainīgo `i` un `k` sākotnējās vērtības.

```
int i=5, k=7;  
cin >> i;  
cout << i << endl;  
cin >> k;  
cout << k << endl;
```

```
abc  
5  
7
```

Piemērs

- Lai turpinātu darbu pēc neveiksmīgas ievades, ir jāveic divas operācijas:
 - objekta *cin* atbloķēšana ar funkciju *clear*,
 - ievadītās simbolu virknes ignorēšana līdz tuvākajam ENTER (funkcija *ignore*).
- Piemērā, mēģinot nolasīt *i*, iestājās kļūda.
- Pēc tam *cin* objekts tiek atbloķēts ar *clear*.
- Starpbuferis "iztīrīts" ar *ignore*, lai ar nākošo komandu *cin >> k* atkal mēģinātu nolasīt skaitli.

Piemērs

- Piemērā, mēģinot nolasīt *i*, iestājas kļūda.
- Pēc tam *cin* objekts tiek atbloķēts ar *clear*.
- Starpbuferis "iztīrīts" ar *ignore*, lai ar nākošo komandu *cin >> k* atkal mēģinātu nolasīt skaitli.

```
int i=5, k=7;  
cin >> i;  
cout << i << endl;  
cin.clear ();  
cin.ignore (256,'\n');  
cin >> k;  
cout << k << endl;
```

```
abc  
5  
99  
99
```

Vesela skaitļa ievadīšana ar kļūdu apstrādi

```
#include <iostream>
using namespace std;
int main ()
{
int i;
cout << "Input integer value: ";
cin >> i;
while (!cin.good())
{
cin.clear ();
cin.ignore (256, '\n');
cout << "Try again: ";
cin >> i;
};
cout << "Success: " << i << endl;
return 0;}
```

```
Input integer value: abc
Try again: +++
Try again: 123
Success: 123
```

cin funkcijas

Funkcijas	Paskaidrojums	Piezīmes
<code>cin.get(char_mainīgais)</code>	No standarta ievada paņem simbolu	<code>cin.get()</code> nolasa arī nākošo simbolu, neizlaižot tukšumus, bet <code>cin>>c</code> ņem līdz tukšumam
<code>cin.ignore()</code>	Nolasa nākošo simbolu un noignorē to	
<code>cin.clear()</code>	Attīra konsoles ievadu	
<code>cin.peek()</code>	Ļauj lietotājam apskatīt nākošo simbolu ievada rindā	
<code>cin.good()</code>	Vai notika pareizs ievads	