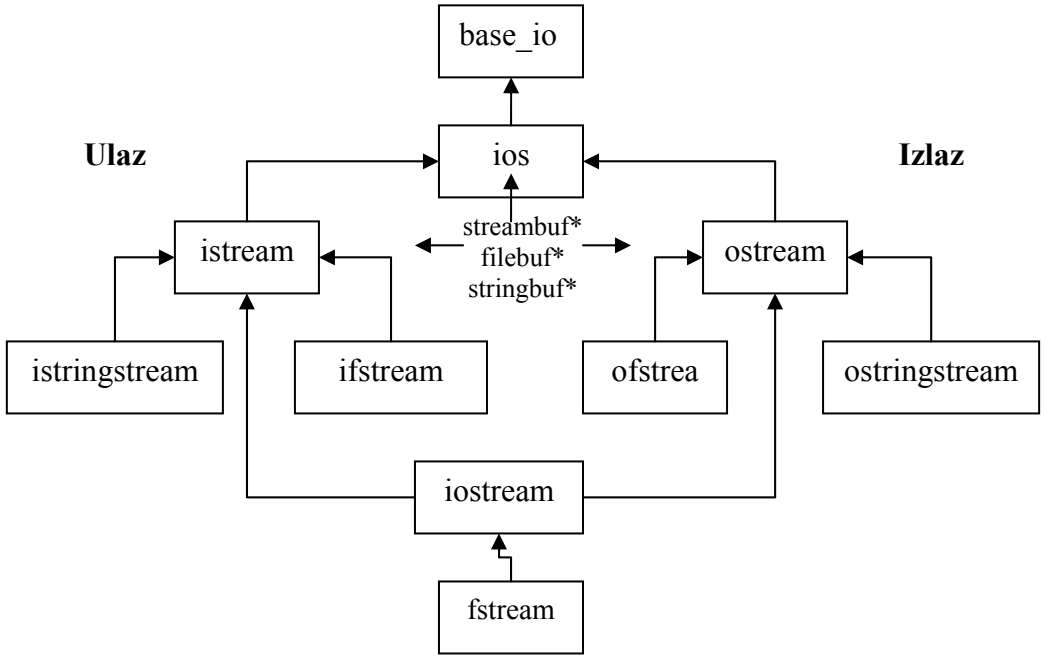
10. Rad s tokovima

Tokovima (streams) nazivamo objekte koji vrše komuniciranje između programa i ulazno-izlaznih uređaja (pr. `cin` i `cout`)

Tokovi se u C++ implementiraju pomoću slijedećih klasa.

- **`streambuf klasa`** služi za implementaciju bafera, a njene članske funkcije služe za punjenje, pražnjenje i čišćenje bafera te za ostale manipulacije s baferom.
- **`base_ios`** je klasa u kojoj su definirane razne konstante
- **`ios`** klasa je temeljna klasa svih ulazno/izlaznih klasa. Ona sadrži streambuf objekt kao člansku varijablu. Nasljeđuje `base_ios` klasu.
- **`istream`** i **`ostream`** klase se izvode iz **`ios`** klase i specijaliziraju za implementaciju ulaznih odnosno izlaznih tokova.
- **`iostream`** klasa koristi **`istream`** i **`ostream`** klase i daje kontrolu pristupa tipkovnici i ispisa na ekranu.
- **`fstream`** klasa služi implementaciji objekata za unos iz datoteke (i**`fstream`**) i ispis u datoteku (**`ofstream`**).
- **`istringstream i ostringstream`** klase omogućuje da se i objekti standardne klase `string` tretiraju kao ulazni/izlazni tokove.



Slika 1. Hijerarhija "iostream" klasa

Napomena: Opisane klase vrijede samo za rad s tekstualnim zapisima u kojima se znakovi kodiraju prema ASCII standardu. U standardnoj ISO biblioteci su predviđene i klase za rad s Unicode kodiranim znakovima.

10.2. Standardni ulazno/izlazni objekti

Svaki C++ program u kojem se uključi zaglavlje <**`iostream`**>, raspolaže s četiri globalna objekta:

- **`cin`** (čitaj "see-in") je objekt klase `istream` koji manipulira standardni ulaz .

- **`cout`** (čitaj "see-out") je objekt klase `ostream` koji manipulira standardni izla

- **`cerr`** (čitaj "see-err") je objekt klase `ostream` koji manipulira standardni izlaz na ekran. Za razliku od **`cout`** objekta, cerr ne koristi baferovani ispis na ekran.

- **`clog`** (čitaj "see-log") je objekt klase `ostream` koji služi za dojavu greške na standardnom izlazu.

10.2.1 Unos podataka

Unos podataka u referirane varijable ili objekte, najčešće se vrši pomoću operatora >>. Ovaj se operator naziva ekstracijski ili "get-to" operator. Primjerice, sa

```
int iVar;
cout << "Unesi cijeli broj: ";
cin >> iVar;
```

vrši se unos cjelobrojne varijable iVar. Ekstracijski operator je preopterećen tako da se pomoću njega može unijeti vrijednost svih standardnih tipova.

Kompilator na temelju deklaracije preopterećenog operatora >> odlučuje koju će funkciju pozvati za unos s tipkovnice. U ovom slučaju bit će pozvana funkcija kojoj je deklaracija oblika;

```
istream& operator>> (iostream &is, int &);
```

Korisnik može preopteretiti ovaj operator za bilo koji korisnički tip podatka.

Ova operatorska funkcija vraća referencu **`istream`** objekta **`cin`**. Zbog toga se može pisati:

```
int var1, var2;
cout << "Unesi dva cijela broja: "
cin >> var1 >> var2;
```

Postupak ekstrakcije je slijedeći. Najprije se izvršava prva ekstrakcija (**`cin>>var1)`**. Pošto ona vraća referencu od **`cin`**, na nju se ponovo može primijeniti ekstrakcijski operator, tj. možemo pisati

```
(cin >> var1) >> var2;
```

Pored ekstrakcijskog operatora >>, **`cin`** objekt raspolaže nizom članskih funkcija koje se mogu koristiti za unos znakova i stringova.

Unos znakova

Za unos znakova se pored ekstracijskog operatora može koristiti članska funkcija get(), i to na dva načina. Prvi je način da se get() koristi bez parametara:

```
char ch;
ch = cin.get();  // 1. način – get() bez parametara
```

Drugi je način da se koristi poziv get() s argumentom tipa char &:

```
char ch;
cin.get(ch);     // 2. način – get() s parametrom tipa char
```

U ovom slučaju funkcija **get(ch)** vraća referencu **cin**, pa se pozivi **get()** mogu višestruko ponavljati. Primjer je dan u programu get.cpp.

```
// Datoteka: get.cpp
// Upotreba get() s parametrom tipa char&
#include <iostream>
using namespace std;
int main()
{
    char a, b, c;
    cout << "Otipkaj tri slova: ";
    cin.get(a).get(b).get(c);
    cout << "a: " << a << "\nb: " << b << "\nc: " << c << endl;
    return 0;
}
```

```
Ispis:
Unesi tri slova: xyz
a: x
b: y
c: z
```

Problem je kod unosa znakova da se i prazno mjesto smatra znakom. Drugi problem je što za unos jednog znaka zapravo treba pritisnuti dvije tipke: tipku znaka i tipku <enter>, kojom završava unos.

Unos stringa sa standardnog ulaza

Za unos stringa također se može koristiti globalni objekt **cin**.

```
char str[80];
cin >> str;     // dobava stringa s tipkovnice u str
cout << str;    // ispis stringa
```

Ako korisnik otipka: **Hello!**
program ispisuje isti tekst: **Hello!**
Ako pak korisnik otkuca: **Hello World!**
program ispisuje samo prvu riječ: **Hello**

Problem je u tome što se unos smatra izvršenim čim se otkuca znak razmaka (‘ ‘). Da bi se mogao izvršiti unos stringova koji sadrže i znak razmaka **istream** klasa raspolaže s članskom funkcijom **get(char *str, int n),** pomoću koje ja moguć unos do maksimalno n-1 znakova, primjerice sa

```
char str[80];
cin.get(str, 80);    // unos stringa s praznim mjestima
cout << str;         // ispis string
```

može se izvršiti unos stringa od maksimalno 79 znakova.

Pokazat ćemo da i ovaj način unosa može biti problematičan. Recimo da želimo unijeti podatke o godini studija u varijablu **godina** i ime studenta u string **ime**. Tada bi napisali:

```
int godina;
char ime[40];
cout << "Unesi godinu studija: ";
cin >> godina;                          // unos broja
cout << "Unesi ime studenta:";
cin.get(ime, 40);                       // unos stringa
```

Kod mnogih kompilatora ne bi se mogao izvršiti unos imena studenta, jer iza unosa broja u ulaznom toku ostaje znak za novu liniju ('\n'). Kada se pozove funkcija **cin.get()** najprije se očita taj znak, a pošto je taj znak ujedno i znak za kraj unosa, ne može se izvršiti unos imena.

Ovaj problem se može izbjeći pomoću istream članske funkcije

```
ignore(int n=1, int delimiter=eof)
```

kojom se zanemaruje slijedećih n znakova do znaka **delimiter**.

```
int godina;
char ime[40];
cout << "Unesi godinu studija: ";
cin >> godina;                    // unos broja
cin.ignore(10, '\n');             // zanemari "novu liniju"
cout << "Unesi ime studenta:";
cin.get(name, 40);                // unos stringa
```

Alternativno **cin.get()** funkciji može se koristiti funkcija **cin.getline(str,n)**. Razlika između ove dvije funkcije je u tome da **getline** skida s toka znak nove linije, iako ga ne umeće u string, odnosno

```
cin.getline(str, maxchar);
```

je ekvivalentno

```
cin.get(str, maxchar);   cin.ignore(1, '\n');
```

10.2.3 Izlazni objekt

Do sada smo koristili operator << za ispis različitih tipova podataka. Ispis također vršimo tako da na izlazne objekte primjenjujemo članske funkcije **`put()`** i **`write()`.**

`ostream& ostream::` **`put(char c)`**	**vrši ispis znaka c.**
`ostream& ostream::` **`write(char const *buffer, int length)`**	**vrši ispis proizvoljnog broja bajte s neke adrese.**

```
cout.put(`H').put(`e').put(`l').put(`l').put(`o').put(`\n');

Ispis je: Hello
```

Funkcija write se uglavnom koristi pri radu s datotekama, a u radu s standardnim izlazom ima jedino smisla ako se koristi za ispis niza znakova.

```
     char str[] = "Vidi ovo!";

     int fullLength = strlen(str);
     int shortLength = fullLength - 4;

     cout.write(str,fullLength) << "\n";
     cout.write(str, shortLength) << "\n";

Ispis je:
Vidi ovo!
Vidi
```

10.3 Dojava greške pri ulazno/izlaznim operacijama

U radu s tokovima, koji su predstavljeni **ios** objektima, može nastati greška i tada je rad s tokom suspendiran. Moguće je dobiti izvještaj o izvršenoj operaciji s tokovima, a u slučaju greške često je moguće obnoviti regularno stanje toka. Stanje nekog toka se interno bilježi u statusnom bajtu (nazovimo ga **_state**) koji se može ispitati sljedećim konstantama:

ios::badbit	**koristi se za označavanje nedozvoljene operacije, primjerice kada se pokušava učitati datoteka na poziciji koja je izvan kraja datoteke.**
ios::eofbit	**koristi se za označavanje stanja kada je dostignut kraj datoteke.**
ios::failbit	**koristi se za označavanje stanja nekorektno izvršene prethodne ulazno/izlazne operacije (nastaje primjerice, kada se dobavlja cijeli broj a korisnik otkuca slova).**
ios::goodbit	**koristi se označavanje stanja korektno izvršene prethodne ulazno/izlazne operacije.**

Ako izraz (**_state & failbit**) daje nenultu vrijednost, to znači da prethodna operacije nije uspjela. Korisnik ne pristupa direktno varijabli **_state** već se koristi sljedećim članskim funkcijama:

int ios::bad()	**funkcija vraća nenultu vrijednost kada se zahtijeva nedozvoljena operacija (tada je postavlja bit određen ios::goodbit konstantom). Primjerice, funkcija bad je definirana sa : int bad() const { return _state & ios::badbit; }**
int ios::eof()	**funkcija vraća nenultu vrijednost kada se dođe do kraja datoteke (EOF) (tada je postavljen ios::eofbit).**
int ios::fail()	**funkcija vraća nenultu vrijednost ako ios::eof() ili ios::bad() vraćaju nenultu vrijednost.**
int ios::good()	**funkcija vraća nenultu vrijednost ako ios::fail() vraća nulu, i obrnuto.**

Primjerice:

```
cin >> x;   // cin je ios objekt
if (cin.good())
      cout << "Unos vrijednosti `x' izvršen uspješno!\n";
```

Ios objekti can se također mogu koristiti kao da vraćaju logičku vrijednost **true** kada je stanje **ios** objekta takvo da **ios::good()** vraća nenultu vrijednost. Zbog toga su dozvoljeni sljedeći iskazi:

```
cin >> x;
if (!cin)
   cout << "Unos vrijednosti `x' nije izvršen uspješno!\n";
cin.clear();
```

10.5. Formatiranje

10.5.1 Manipulatori i članske funkcije

Za formatiranje ispisa koriste se manipulatori i/ili članske funkcije. Manipulatori se koriste uz operatore << i >>. Primjerice, do sada smo koristili manipulator **endl** za kontrolu prijelaza u novu liniju Uloga tog manipulatora je, pored toga, da se i svi znakovi, koji su preostali u izlaznom baferu, ispišu na ekranu.

Operaciju pražnjenja bafera se može i eksplicitno zadati pomoću manipulatora imena flush. Naredba

```
cout << x << endl;
```

je ekvivalentna naredbi

```
cout << x << '\n'<<flush;
```

Kada se manipulator **flush** koristi na ulaznim objektima tada se odbacuju svi znakovi koji su preostali u baferu.

```
// Ugadjanje sirine ispisa
       cout << "Start >";
       cout.width(25);
       cout << 123 << "< End\n";

       cout << "Start >";
       cout.width(25);
       cout << 123<< "< Next >";
       cout << 456 << "< End\n";

       cout << "Start >";
       cout.width(6);
       cout << 123456 << "< End\n";
```

```
Output:
Start >                  123< End
Start >                  123< Next >456< End
Start >123456< End
```

Uočite da funkcija width() djeluje samo na jedan pristup izlaznom toku. Nakon toga ponovo vrijedi predodređeni format ispisa.

Preodređeni format

Skalarni cjelobrojni tipovi (osim **char** i **unsigned char**) imaju decimalni format,

Pokazivači (osim **char*** i **unsigned char***) se zapisuju u heksadecimalnoj notaciji ,

Realni (**float** i **double**) tipovu se zapisuju s 6 znamenki iza decimalne točke (kažemo da je preciznot ispisa na 6 znamenki).

Znakovi (**char** i **unsigned char**) se ispisuju svojim ASCII znakom, a **char*** i **unsigned char*** se tretiraju kao pokazivači ACSIIZ stringa.

Pri unošenju cijelih brojeva dozvoljeno je je pisati razmak prije prve znamenka. Predznak (+, -) se mora pisati bez razmaka. Unos završava s znakom koji nije znamenka. Slično vrijedi i za realne brojeve, ali tada su dozvoljeni i znakovi decimalne točke i eksponeta (E).

U većini slučajeva ovaj predodređeni način formatiranja je dovoljan.

10.5.2 Kako se kontrolira formatiranje

Svakom iostream objektu pripada jedna varijabla stanja (tipa int) u kojoj se bilježi trenutni format. Pojedini bit u ovoj varijabli (zastavica) označava stanje formata. Za postavljanje ovih zastavica u klasi `base_ios` deklarirano je više konstanti, primjerice, sa

```
enum
{  skipws=01,                          // preskok znaka razmaka pri unosu
   left=02, right=04, internal=010,    // poravnanje ispisa
   dec=020, oct=040, hex=0100,         // baza notacije ispisa
   showbase=0200, showpoint=0400,
   uppercase=01000, showpos=02000,     // modifikatori
   scientific=04000, fixed=010000      // oblik ispisa realnih brojeva
} ;
```

Format se može postaviti s članskom funkcijom setf(), a njegovo stanje se dobavlja članskom funkcijom flag().

Primjerice, **ios::skipws** (ili base_ios:skipws) je konstanta pomoću koje se postavlja zastavica kojom se kontrolira da li se preskače razmak pri unosu vrijednosti (eng. skip white space).

```
char c ;
cin.setf(0,ios::skipws);                  // isključi preskok razmaka
cin >> c ;
cin.setf(ios::skipws, ios::skipws) ;      // uključi preskok razmaka
```

Prvi argument označava vrijednost koja se postavlja, a drugi argument (maska) označava koja se zastavica postavlja.

U istu svrhu se mogu koristiti i manipulatori (koji su deklarirani u < **iomanip>).** Gornji primjer se može pomoću manipulatora zapisati u obliku:

```
cin  >> resetiosflags(ios::skipws)
     >> c
     >> setiosflags(ios::skipws) ;
```

resetiosflags postavlja zastavicu određenu s argumentom na vrijednost 0, **setiosflags** postavlja zastavicu na vrijednost 1).

Konstante za postavljene zastavica

`ios::adjustfield`:	*mask value* used in combination with a flag setting defining the way values are adjusted in wide fields (`ios::left`, `ios::right`, `ios::internal`).
`ios::basefield`	*mask value* used in combination with a flag setting the radix of integral values to output (`ios::dec`, `ios::hex` or `ios::oct`).
`ios::boolalpha`	to display boolean values as text, using the text ``true'' for the `true` logical value, and the string ``false'' for the `false` logical value. By default this flag is not set. Corresponding manipulators: `ios::boolalpha`, `ios::noboolalpha`.
`ios::dec`	to read and display integral values as decimal (i.e., radix 10) values. This is the default. With `setf()` the mask value `ios::basefield` must be provided. Corresponding manipulator: `ios::dec`.
`ios::fixed`	to display real values in a fixed notation (e.g., 12.25), as opposed to displaying values in a scientific notation. With `setf()` the mask value `ios::floatfield` must be provided. Corresponding manipulator: `ios::fixed`.
`ios::floatfield`	mask value used in combination with a flag setting the way real numbers are displayed (`ios::fixed` or `ios::scientific`).
`ios::hex`	to read and display integral values as hexadecimal values (i.e., radix 16) values. With `setf()` the mask value `ios::basefield` must be provided. Corresponding manipulator: `ios::hex`.
`ios::internal`	to add fill characters (blanks by default) between the minus sign of negative numbers and the value itself. With `setf()` the mask value `adjustfield` must be provided. Corresponding manipulator: `ios::internal`.
`ios::left`	to left-adjust (integral) values in fields that are wider than needed to display the values. By default values are right-adjusted (see below). With `setf()` the mask value `adjustfield` must be provided. Corresponding manipulator: `ios::left`.
`ios::oct`	to display integral values as octal values (i.e., radix 8) values. With `setf()` the mask value `ios::basefield` must be provided. Corresponding manipulator: `ios::oct`.
`ios::right`	to right-adjust (integral) values in fields that are wider than needed to display the values. This is the default adjustment. With `setf()` the mask value `adjustfield` must be provided. Corresponding manipulator: `ios::right`.
`ios::scientific`	to display real values in *scientific notation* (e.g., 1.24e+03). With `setf()` the mask value `ios::floatfield` must be provided. Corresponding manipulator: `ios::scientific`.
`ios::showbase`	to display the numeric base of integral values. With hexadecimal values the `0x` prefix is used,

	with octal values the prefix `0`. For the (default) decimal value no particular prefix is used. Corresponding manipulators: `ios::showbase` and `ios::noshowbase`

`ios::showpoint`	display a trailing decimal point and trailing decimal zeros when real numbers are displayed. When this flag is set, an insertion like:
	
	`cout << 16.0 << ", " << 16.1 << ", " << 16 << endl;`
	
	could result in:
	
	`16.0000, 16.1000, 16`
	
	Note that the last `16` is an integral rather than a real number, and is not given a decimal point: `ios::showpoint` has no effect here. If `ios::showpoint` is not used, then trailing zeros are discarded. If the decimal part is zero, then the decimal point is discarded as well. Corresponding manipulator: `ios::showpoint`.

`ios::showpos`	display a `+` character with positive values. Corresponding manipulator: `ios::showpos`.
`ios::skipws`	used for extracting information from streams. When this flag is set (which is the default) leading white space characters (blanks, tabs, newlines, etc.) are skipped when a value is extracted from a stream. If the flag is not set, leading white space characters are not skipped.
`ios::unitbuf`	flush the stream after each output operation.
`ios::uppercase`	use capital letters in the representation of (hexadecimal or scientifically formatted) values.

Članske funkcije za kontrolu formatiranja

`ios::setf(` `fmtflags flags)`	returns the *previous* set of *all* flags, and sets one or more formatting flags (using the bitwise operator `|` to combine multiple flags). Other flags are not affected. Corresponding manipulators: `ios::setiosflags` and `ios::resetiosflags`

`ios::setf(`
`fmtflags flags,`
`fmtflags mask)`
	returns the *previous* set of *all* flags, clears all flags mentioned in `mask`, and sets the flags specified in `flags`. Well-known mask values are `ios::adjustfield`, `ios::basefield` and `ios::floatfield`. For example:
	
	`setf(ios::left, ios::adjustfield)` is used to left-adjust wide values in their field. (alternatively, `ios::right` and `ios::internal` can be used).
	
	`setf(ios::hex, ios::basefield)` is used to activate the hexadecimal representation of integral values (alternatively, `ios::dec` and `ios::oct` can be used).

	`setf(ios::fixed, ios::floatfield)` is used to activate the fixed value representation of real values (alternatively, `ios::scientific` can be used).
`ios::unsetf(fmtflags flags)`	returns the *previous* set of *all* flags, and clears the specified formatting flags (leaving the remaining flags unaltered). The unsetting of an active default flag (e.g., `cout.unsetf(ios::dec)`) has no effect.
`ios::flags() const`	returns the current collection of flags controlling the format state of the stream for which the member function is called. To inspect a particular flag, use the binairy and operator, e.g., if (cout.flags() & ios::hex) { // hexadecimal output of integral values }
`ios::flags(fmtflags flagset)`	returns the *previous* set of flags, and defines the current set of flags as `flagset`, defined by a combination of formatting flags, combined by the binary or operator.
`ios::width() const`:	returns (as `int`) the current output field width (the number of characters to write for numerical values on the next insertion operation). Default: 0, meaning `as many characters as needed to write the value'. Corresponding manipulator: `ios::setw()`.
`ios::width(int nchars)`	returns (as `int`) the previously used output field width, redefines the value to `nchars` for the next insertion operation. Note that the field width is reset to 0 after every insertion operation, and that `width()` currently has no effect on text-values like `char *` or `string` values.
`ios &copyfmt(ios &obj)`	This member function copies all format definitions from `obj` to the current `ios` object.
`ios::fill() const`	returns (as `char`) the current padding character. By default, this is the blank space.
`ios::fill(char padding)`	redefines the padding character. Returns (as `char`) the *previous* padding character. Corresponding manipulator: `ios::setfill()`.
`ios::precision() const`	returns (as `int`) the number of significant digits used for outputting real values (default: 6).
`ios::precision(int signif)`	redefines the number of significant digits used for outputting real values, returns (as `int`) the previously used number of significant digits. Corresponding manipulator: `ios::setprecision()`.

Korištenje manipulatora

Formatiranje se češće zadaje pomoću manipulatora. Manipulatori su funkcije, koje se uglavnom koriste bez argumenata. Ako se koriste manipulatori koji koriste argumente, tada se mora koristiti zaglajavlje `<iomanip>`.

`boolalpha`	This manipulator will set the `ios::boolalpha` flag.

`dec`	This manipulator enforces the display and reading of integral numbers in decimal format. This is the default conversion. The conversion is applied to values inserted into the stream after processing the manipulators. For example (see also `ios::hex` and `ios::oct`, below):
	
	`cout << 16 << ", "`
	`  << hex << 16 << ", "`
	`  << oct << 16;`
	`// produces the output:`
	`  16, 10, 20`

`endl`	This manipulator will insert a newline character into an output buffer and will flush the buffer thereafter.
`ends`	This manipulator will insert a string termination character into an output buffer.
`fixed`	This manipulator will set the `ios::fixed` flag.
`flush`	This manipulator will flush an output buffer.
`hex`	This manipulator enforces the display and reading of integral numbers in hexadecimal format.
`internal`	This manipulator will set the `ios::internal` flag.
`left`	This manipulator will align values to the left in wide fields.
`noboolalpha`	This manipulator will clear the `ios::boolalpha` flag.
`noshowpoint`	This manipulator will clear the `ios::showpoint` flag.
`noshowpos`	This manipulator will clear the `ios::showpos` flag.
`noshowbase`	This manipulator will clear the `ios::showbase` flag.
`noskipws`	This manipulator will clear the `ios::skipws` flag.
`nounitbuf`	This manipulator will stop flushing an output stream after each write operation. Now the stream is flushed at a `flush`, `endl`, `unitbuf` or when it is closed.
`nouppercase`	This manipulator will clear the `ios::uppercase` flag.
`oct`	This manipulator enforces the display and reading of integral numbers in octal format.
`resetiosflags(flags)`	This manipulator calls `ios::resetf(flags)` to clear the indicated flag values.
`right`	This manipulator will align values to the right in wide fields.
`scientific`	This manipulator will set the `ios::scientific` flag.

`setbase(int b)`	This manipulator can be used to display integral values using the base 8, 10 or 16. It can be used as an alternative to `oct, dec, hex` in situations where the base of integral values is parameterized.
`setfill(int ch)`	This manipulator defines the filling character in situations where the values of numbers are too small to fill the width that is used to display these values. By default the blank space is used.
`setiosflags(flags)`	This manipulator calls `ios::setf(flags)` to set the indicated flag values.
`setprecision(int width)`	This manipulator will set the precision in which a `float` or `double` is displayed.
`setw(int width)`	This manipulator expects as its argument the width of the field that is inserted or extracted next. `setw()` is valid *only* for the next field. It does *not* act like e.g., `hex` which changes the general state of the output stream for displaying numbers. A nice feature is that a long string appearing at `cin` is split into substrings of at most `sizeof(array) - 1` characters, and that an `ASCII-Z` character is automatically appended.
`showbase`	This manipulator will set the `ios::showbase` flag.
`showpoint`	This manipulator will set the `ios::showpoint` flag.
`showpos`	This manipulator will set the `ios::showpos` flag.
`skipws`	This manipulator will set the `ios::skipws` flag.
`unitbuf`	This manipulator will flush an output stream after each write operation.
`uppercase`	This manipulator will set the `ios::uppercase` flag.
`ws`	This manipulator will remove all whitespace characters that are available at the current read-position of an input buffer.

Napomena: Iako su manipulatori realizirani kao globalne funkcije, oni se koriste na nestandardan način, tj. iz imena manipulatorara se na zapisuju oble zagrada. Znamo da ime funkcije napisano bez zagrada predstavlja adresu. Postavlja se pitanje kako se onda poziva ove funkcije. To je omogućeno posebnim mehanizmom koji je implementiran u iostream klasama, da se za objekte, čija je vrijednost adresa funkcije manipulatora, pozove ta funkcija.

10.5.3 Primjeri korištenja članskih funkcija i manipulatora

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

int main()
{
    const int number = 185;
    cout << "Broj je " << number << endl;

    cout << "Broj je  " << hex <<  number << endl;

    cout.setf(ios::showbase);
    cout << "Broj je " << hex <<  number << endl;

    cout << "Broj je " ;
    cout.width(10);
    cout << hex << number << endl;

    cout << "Broj je " ;
    cout.width(10);
    cout.setf(ios::left);
    cout << hex << number << endl;

    cout << "Broj je " ;
    cout.width(10);
    cout.setf(ios::internal);
    cout << hex << number << endl;

    cout << "Broj je " << setw(10) << hex << number <<
endl;
     return 0;
}
```

Ispis je:

```
Broj je 185
Broj je  b9
Broj je 0xb9
Broj je       0xb9
Broj je 0xb9
Broj je       0xb9
Broj je       0xb9
```

Formatiranje floating point brojeva

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

// formatiranje pomoću manipulatora
int main()
{
    const double PI = acos( - 1);
    const int MAX = 10;
   const int TAB = 15;
   int k;
   cout << "predodredjeni format " << PI
        << ", sa setprecision(4), "
        << setprecision(4) << PI
        << endl;
    cout <<"fixed/scientific realni brojevi\n";

   for(k=0; k < MAX; k++)
   {
     cout << left << "pre. " << k << "\t"
          << setprecision(k)
          << setw(TAB)
          << fixed << PI << scientific << "\t" << PI
          << endl;
   }

   return 0;
}
```

```
predodredjeni format 3.14159, sa
setprecision(4), 3.142
fixed/scientific realni brojevi:
pre. 0  3               3.141593e+000
pre. 1  3.1             3.1e+000
pre. 2  3.14            3.14e+000
pre. 3  3.142           3.142e+000
pre. 4  3.1416          3.1416e+000
pre. 5  3.14159         3.14159e+000
pre. 6  3.141593        3.141593e+000
pre. 7  3.1415927
3.1415927e+000
pre. 8  3.14159265
3.14159265e+000
pre. 9  3.141592654
3.141592654e+000
```

Manipulator `setw` vrijedi samo u slijedećem ispisu, a ostali vrijede dok ih se ne promijeni.