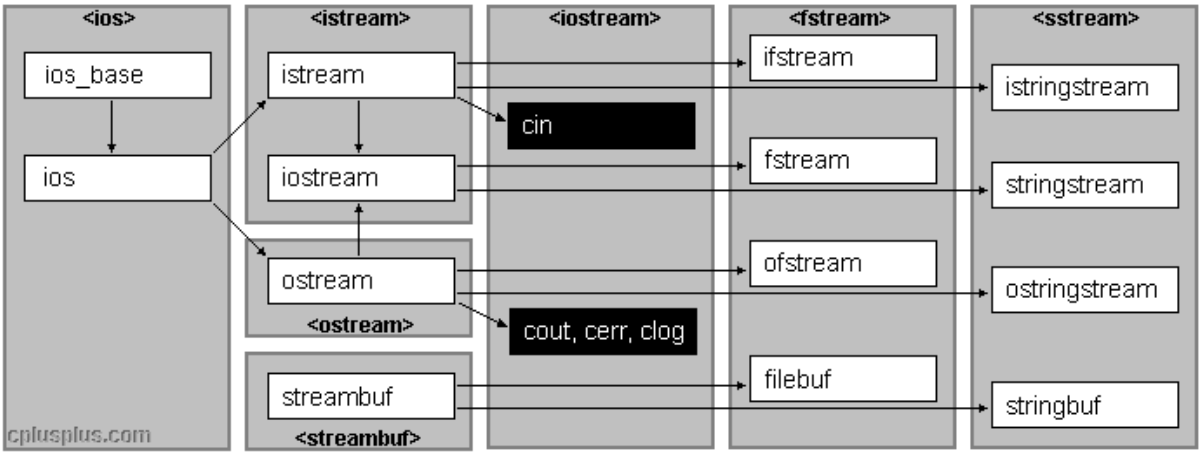
iostream

Jazyk C++ se kouká na standardní vstup (klávesnice na PC) a standardní výstup (konzole na PC) jako na *datové toky (**streams**)*. To je abstrakce nad posíláním dat (output stream, knihovna `ostream`) a přijímáním dat (input stream, knihovna `istream`). Vlastnosti obou knihoven jsou sloučeny do knihovny `iostream`.

Datové toky nemusí být pouze standardní: jejich zdrojem může být také soubor (třída fstream z knihovny `fstream` nebo řetězec (třída stringstream z knihovny `sstream`). Datové toky jsou realizovány pomocí tříd, jejichž závislosti vystihuje obrázek z http://cplusplus.com/reference/iolibrary/



Na černém pozadí jsou **objekty** těchto tříd spojené se standardním vstupem (`cin`), standardním výstupem (`cout`) a standardními chybovými výstupy (`cerr`, `clog`), přičemž `cerr` je ve výchozím stavu přesměrován na standardní výstup.

`istream` definuje operátor výběru (*extract operator*) `>>`, který vybírá ze vstupního toku data a přesouvá je do proměnných. Tento operátor je přetížen (tj. má více definic, kompilátor vybere tu, kde vyhovují typy argumentů) na všechny číselné typy a vrací refereci na svůj první argument (`istream&`), takže je možné jej řetězit (*chaining*). Můžeme tedy psát:

```
using namespace std;

int i;
double d;

cin >> i; // přetížení na int
cin >> d; // přetížení na double
cin >> i >> d; // řetězení, provede se jako (cin >> i) >> d
```

Podobně `ostream` definuje operátor vkládání (*insert operator*) `<<`, který na standardní výstup vkládá data z proměnných. Můžeme tedy psát:

```
using namespace std;

int i = 42;
double d = 3.14;

cout << "odpoved je " << i << ", pi=" << d << endl;
```

Kde `endl` je konstanta konce řádku. Zní to dobře, ale `iostream` má velmi komplikované odpovědi na velmi jednoduché otázky. Např.

Problém – jak omezit délku vstupu?

Za pomocí `scanf` to lze snadno:

```
char buffer[16];
scanf("%15s%n", buffer); // načti max 15 znaků
while(getchar()!='\n');  // vyčisti vstupní frontu
```

A jak na to pomocí operátoru `>>`? Nelze. Řešení 1: použít třídu string:

```
string str;
cin >> str; // načti vše do str (může zabrat zbytečně mnoho paměti)
str.resize(15); // ořízni na 15. pozici
cout << str;
```

Řešení 2: použít metodu `getline`:

```
char buffer[16];
cin.getline(buffer, 16);
if(cin.fail()) {          // na vstupu jsou stále data
  cin.clear();            // vyčisti příznak failbit
  cin.ignore(8192, '\n'); // zahoď všechny další znaky na vstupu
}
```

S podobnými neintuitivními věcmi (nutno vědět, že existují nějaké chybové příznaky), se potkáváme kdykoliv máme řešit načítání čehokoliv složitějšího než základní datové typy (např. čti včetně mezer, čti až do středníku apod.).

Problém – jak načíst číslo?

```
int n;
if(scanf("%d", &n)) printf("input: %d\n", n);
else puts("chyba");
```

Pomocí iostream:

```
int n;
cin >> n;
if(cin.fail()) {
  cin.clear();
  cin.ignore(8192, '\n');
  cout << "chyba";
}
else cout << n;
```

Pokud tedy píšete nízkoúrovňový program, zůstaňte raději u `cstdio`. Přednosti `iostream` vyniknout při implementaci vlastních typů, ale o tom až příště.