

Programmering i C++

EDA623

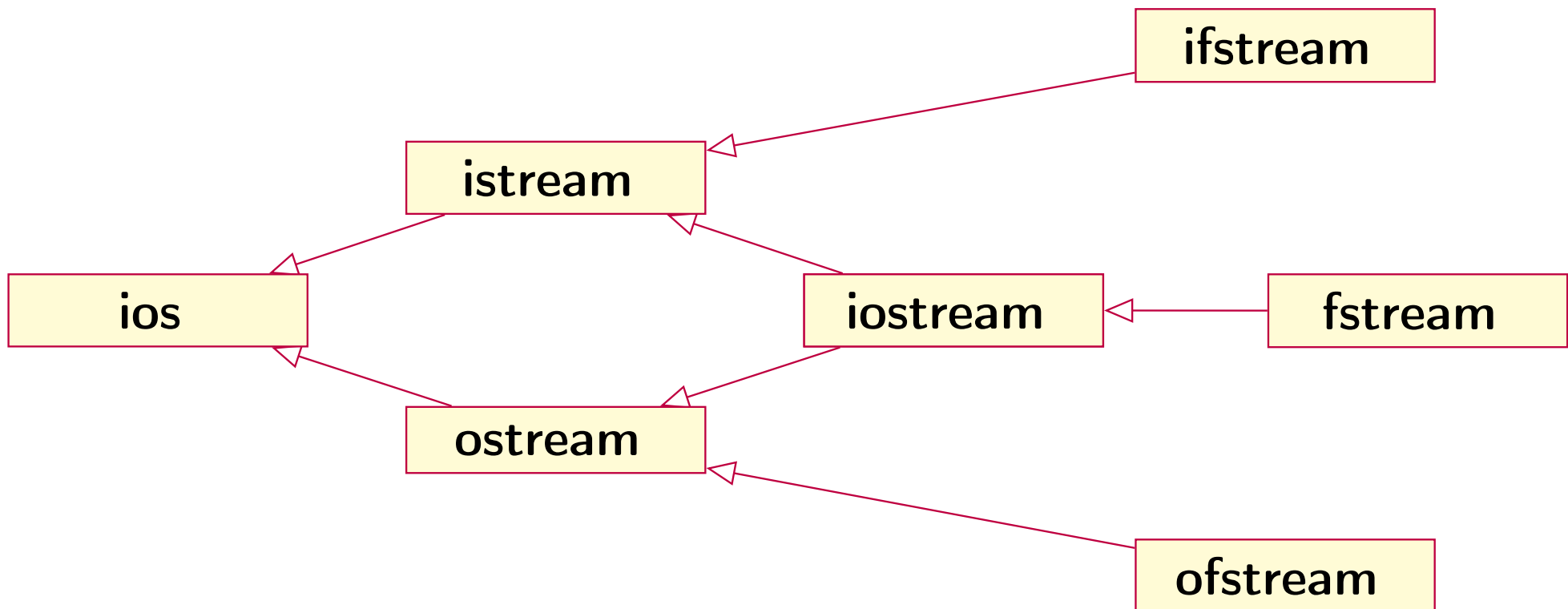
Strömmar och filer

Innehåll

- Klassen `ios`
- Läsning av strömmar
- Utskrift av strömmar
- Koppling av filer till strömmar
- Direktaccess

- Ström (stream) = Följd av tecken (bytes)
- Grundläggande klass för strömmar: ios

Klassträd för strömmar



- Fördefinierade strömmar deklarerade i `<iostream>`:
`cin`, `cout`, `cerr`, och `clog`
- Klasser deklarerade i inkluderingsfilen: `<fstream>`:
`ifstream`, `ofstream`, `fstream`
- Flaggor definierade i `ios`, vilka beskriver en ströms tillstånd:
 - `failbit` Senaste operationen misslyckades
 - `eofbit` Ett filslut påträffades vid senaste operationen
 - `badbit` Ett allvarligare fel av intern art har inträffat

Medlemsfunktioner i klassen ios:

```
void clear();    Slår av alla tillståndsflaggorna
bool good();    Ger true om alla flaggor false
bool fail();    Ger true om failbit el. badbit satt
bool eof();    Ger true om eofbit är satt
bool bad();    Ger true om badbit är satt
bool operator!();    Ger resultatet fail()
```

Inläsning tecken för tecken

```
char c;
// Felaktigt sätt:
while (!cin.eof()) {
    cin.get(c);
    //...
} // Läser 2 tecken för mycket (LF och EOF)

// Korrekt sätt:
while (cin.get(c)) {
    //...
} // Läser rätt antal tecken
```

Strömmar från klassen `istream` eller från subclass till denna

- Oformaterad inmatning: Läser data från strömmen utan att konvertera till annat format
 - Görs via medlemsfunktioner
- Formaterad inmatning: Data från strömmen görs om till annat format
 - Görs med operatorn `>>`

Formaterad inmatning med >>

```
// Kodavsnitt
string avr_ort;
int dag, manad;
cout << "Ange avreseort, dag och månad: ";
cin >> avr_ort >> dag >> manad;
cout << "Avresa från " << avr_ort
    << " den " << dag << '/' << manad << endl;
```

```
// In- och utmatning
Ange avreseort, dag och månad: Lund 24 10
Avresa från Lund den 24/10
```


Manipulatorer vid inmatning:

<code>setw(n)</code>	Max-antalet tecken i inläsningssträngen
<code>ws</code>	Hoppa fram till nästa icke-vita tecken
<code>skipws</code>	Hoppa över inl. vita tecken vid anv av >>
<code>noskipws</code>	Hoppa ej över inl. vita tecken vid anv av >>
<code>dec</code>	Tolka följande heltal som decimalt
<code>oct</code>	Tolka följande heltal som oktalt
<code>hex</code>	Tolka följande heltal som hexadecimalt
<code>boolalpha</code>	Indata för <code>bool</code> på formen <code>false</code> / <code>true</code>
<code>noboolalpha</code>	Indata för <code>bool</code> på formen 0 / 1

Formaterad inmatning med >> och manipulatorer

```
// Kodavsnitt
#include <iomanip>

int i, j, k;
cout << "Ange tre heltal: "
cin >> oct >> i >> hex >> j >> k;
cout << i << " " << j << " " << k << endl;

// In- och utmatning
Ange tre heltal:
12 34 56
10 52 56
```

Oformaterad inmatning med medlemsfunktioner

<code>gcount()</code>	Anger antalet tecken vid senaste inläsn.
<code>get()</code>	Läser in och returnerar nästa tecken
<code>get(c)</code>	Läser in tecken till <code>c</code>
<code>getline(s, n, t)</code>	Läser <code>n</code> tecken till <code>s</code> med <code>t</code> som radseparator
<code>get(s, n, t)</code>	Som <code>getline</code> men sep. <code>t</code> läses ej
<code>read(s, n)</code>	Läser in <code>n</code> st tecken till <code>s</code>
<code>ignore(s, t)</code>	Hoppar över <code>n</code> st tecken vid inl till <code>s</code>
<code>peek()</code>	Returnerar nästa tecken (som förblir oläst)
<code>putback(c)</code>	Lägger tillbaka <code>c</code> i strömmen
<code>unget()</code>	Lägger tillbaka senast lästa tecken

Formaterad utmatning med <<

```
cout << uppercase << "hej svejs" << endl;  
// ger utskriften
```

hej svejs

```
cout << uppercase << scientific << 123456789.0 << endl;  
// ger utskriften
```

1234568E+008

Manipulatorer vid utmatning:

<code>uppercase</code>	Ger stort E vid flyttalsutskrift
<code>fixed</code>	Utskrift med fixnotation
<code>scientific</code>	Utskrift med flytnotation
<code>hex</code>	Utskrift som hexadecimalt tal
<code>oct</code>	Utskrift som oktalt tal
<code>dec</code>	Utskrift som decimalt tal
<code>setw(n)</code>	Sätter minimalt antal positioner
<code>setfill(c)</code>	Anger tecken för utfyllnad (padding)
<code>setprecision(n)</code>	Sätter antalet decimaler (om <code>fixed</code>) eller antalet sign. siffror (om <code>scientific</code>)
<code>setbase(n)</code>	<code>setbase(16)</code> , <code>setbase(8)</code> osv
<code>flush</code>	Tömmer utskriftsbufferten
<code>endl</code>	Lägger in radslut i strömmen

Oformaterad utmatning med medlemsfunktioner

<code>put(c)</code>	Skriv ut tecknet <code>c</code> till strömmen
<code>write(s, n)</code>	Skriv ut <code>n</code> tecken från fältet <code>s</code>
<code>flush()</code>	Töm utskriftsbufferten

Koppling av filer till strömmar

Öppnande av fil för läsning

```
ifstream infile("infilen.txt");  
  
// Alt.  
ifstream infile;  
infile.open("infilen.txt");
```

Öppnande av fil för skrivning

```
ofstream outfile("outfilen.txt");  
  
// Alt.  
ofstream outfile;  
outfile.open("outfilen.txt");
```

Stängning av fil

```
infil.close();  
utfil.close();
```

Ofta behövs inte `close` användas eftersom destruktörerna för `ifstream` och `ofstream` automatiskt anropas då den associerade strömmen termineras

Koppling av filer till strömmar

Kopiering av fil

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

main (int argc, char* argv[]) {
    if (argc != 3) {
        cout << "Syntax: " << argv[0]
             << " from_file to_file" << endl;
    }
    char c;
    ifstream f1(argv[1], ios::binary); //Binärfil
    ofstream f2(argv[2], ios::binary);

    while (f1.get(c)) // Snabbare sätt (via filbuffert):
        f2.put(c);    // f2 << f1.rdbuf();
}
```

Koppling av filer till strömmar

Filflaggor i klassen `ios`

<code>in</code>	Filen skall existera och vara läsbar.
<code>out</code>	Om filen existerar skall den skrivas över. Om filen inte finns skall en ny skrivbar fil skapas.
<code>app</code>	Om filen existerar skall skrivning läggas till i slutet. Om filen inte finns skall en ny skrivbar fil skapas.
<code>trunc</code>	Om filen redan finns skall den skrivas över
<code>ate</code>	Efter öppning flyttas filpekaren till slutet av filen
<code>binary</code>	Filen skall hanteras som en binärfil

Kombination av flaggor med operator | (bitvis eller)

```
ofstream filen("fil.dat",ios::trunc|ios::binary);
```

Direkt indexering av filposition där index är av typen `streampos` (heltalstyp)

<code>is.tellg()</code>	Ger aktuell position i inströmmen <code>is</code>
<code>os.tellp()</code>	Ger aktuell position i utströmmen <code>os</code>
<code>is.seekg(pos)</code>	Sätter aktuell position i <code>is</code> resp. <code>os</code>
<code>os.seekp(pos)</code>	till <code>pos</code> (av typen <code>streampos</code>)
<code>is.seekg(off, dir)</code>	Sätter positionen till <code>off+dir</code> där
<code>os.seekp(off, dir)</code>	<code>off</code> är en (ev neg.) offset och <code>dir</code> är
	<code>ios::beg</code> (start), <code>ios::end</code> (slut) eller
	<code>ios::cur</code> (akt. pos.)