EDAF30 – Programmering i C++

4. Standard-containers. Strömmar och filer

Sven Gestegård Robertz

Datavetenskap, LTH

2017

Innehåll

Läsanvisningar

Referenser till relaterade avsnitt i Lippman

Sekvenser	3.3, 9.1, 9.2, 9.3.1 – 9.3.3
Iteratorer	3.4, 9.2.1
Associativa containers	11.1–11.3.2
Strömmar och filer	Kapitel 8
Formatterad I/O	17.5.1
Oformatterad I/O	17.5.2

Containerklasser – Klassificering

Sekvenser (homogena)

- `vector<T>`
- `deque<T>`
- `list<T>`

Associativa containers (finns även *unordered*)

- `map<K,V>`, `multimap<K,V>`
- `set<T>`, `multiset<T>`

Heterogena sekvenser (inte "containers")

- `tuple<T1, T2, ...>`
- `pair<T1,T2>`

Klasserna vector och deque

Operationer i klassen vector

```
v.clear(), v.size(), v.empty()
v.push_back(), v.pop_back()
v.front(), v.back(), v.at(i), v[i]
v.assign(), v.insert(), v.emplace()
v.resize(), v.reserve()
```

Ytterligare operationer i klassen deque

```
d.push_front(), d.pop_front()
```

Klasserna vector och deque
Konstruktorer och funktionen assign

Konstruktor och assign finns i tre varianter:

- *fill*: n stycken element med samma värde

```
void assign (size_type n, const value_type& val);
```

- *initializer list*

```
void assign (initializer_list<value_type> il);
```

- *range*: kopierar objekten i intervallet $[first, last)$ (d v s från first till last, exkl last)

```
template <class InputIterator>
void assign (InputIterator first, InputIterator last);
```

Klasserna vector och deque
Medlemsfunktionen assign, exempel

```
vector<int> v;
int a[]{0,1,2,3,4,5,6,7,8,9};

v.assign(3,1);
print_seq(v);
```
length = 3: [1][1][1]

```
v.assign({11,13,15,17});
print_seq(v);
```
length = 4: [11][13][15][17]

```
v.assign(a, a+5);
print_seq(v);
```
length = 5: [0][1][2][3][4]

```
std::deque<int> d;
d.assign(v.begin(), v.end());
print_seq(d);
```
length = 5: [0][1][2][3][4]

Exempel på iteratorer

Klasserna vector och deque
Medlemsfunktioner push och pop

push lägger till ett element, storleken ökar

pop tar bort ett element, storleken minskar

*_back opererar på slutet, finns i båda

```
void push_back (const value_type& val);    //copy
void pop_back();
```

bara i deque: *_front

```
void push_front (const value_type& val);   //copy
void pop_front();
```

Iteratorer

Iterator

”Pekarliknande” variabel som används för att genomlöpa en struktur (datasamling)

Exempel

```
vector<double> v(4);

vector<double>::iterator it;
for (it=v.begin(); it != v.end(); ++it)
    *it = 0;

for (auto it=v.begin(); it != v.end(); ++it) // med auto (C++11)
    *it = 0;
for (double &e : v)  // Ekvivalent i C++11
    e = 0;
```

Iteratorer

Funktioner som returnerar iteratorer och som finns i alla containerklasser + klassen `string`

`begin()`	ger en iterator som pekar på det första elementet
`end()`	ger en iterator som pekar på ett tänkt element *efter* det sista elementet
`rbegin()`	ger en reverserad iterator som pekar på det sista elementet
`rend()`	ger en reverserad iterator som pekar på ett tänkt element *före* det första elementet

samt:

```
cbegin()      rcbegin()
cend()        rcend()
```

Iteratorer

Operationer på en sekvens med iteratorer som parametrar (iteratorintervallet `[i,j)` betecknar intervallet från och med `i` till och med positionen före `j`)

sekv<typ> s(i,j); konstruktor, s initeras med `[i,j)`
s.assign(i,j); s = elementen i intervallet `[i,j)`
...
s.insert(p,e); sätter in värdet e i positionen (iterator) p
s.insert(p,n,e); sätter in n stycken e i positionen p
s.insert(p,i,j); sätter in elementen i `[i,j)` i pos. p

s.erase(p); tar bort elementet i position p från s
s.erase(p,p2); tar bort elementen i `[p,p2)`från s

Iteratorer
Exempel: vector::assign, vector::insert och vector::erase

```
int a[] {1,2,3,4};
vector<int> v;
v.assign(a, a+4);
print_seq(v);
```

```
length = 4: [1][2][3][4]
```

```
v.insert(v.begin()+2, 3, 9);
print_seq(v);
```

```
length = 7: [1][2][9][9][9][3][4]
```

```
v.erase(v.begin()+5);
print_seq(v);
```

```
length = 6: [1][2][9][9][9][4]
```

```
v.erase(v.begin(),v.begin()+2);
print_seq(v);
```

```
length = 4: [9][9][9][4]
```

Avbildningar och mängder

Associativa containers

- Tabeller med söknycklar – t.ex. telefonlista med 2 kolumner (namn, telnr) där namnet utgör söknyckel
- Implementering i form av standardklasser

`map<Nyckel,Värde>`	Varje nyckel förekommer precis en gång
`multimap<Nyckel,Värde>`	En nyckel kan förekomma mer än en gång
`set<Nyckel>`	Varje nyckel förekommer precis en gång
`multiset<Nyckel>`	En nyckel kan förekomma mer än en gång

`set` *är i princip en* `map` *utan värden.*

Avbildningar och mängder

<set>: std::set

```
void test_set()
{
    std::set<int> ints{1,3,7};

    ints.insert(5);

    for(auto x : ints) {
        cout << x << " ";
    }
    cout << endl;

    auto has_one = ints.find(1);

    if(has_one != ints.end()){
        cout << "one is in the set\n";
    } else {
        cout << "one is not in the set\n";
    }
}
```

```
1 3 5 7
one is in the set
```

Eller

```
if(ints.count(1))
```

Avbildningar och mängder

<map>: std::map

```
map<string, int> msi;
msi.insert(make_pair("Kalle", 1));
msi.emplace("Lisa", 2);
msi["Kim"]= 5;

for(auto& a: msi) {
    cout << a.first << " : " << a.second << endl;
}

cout << "Lisa --> " << msi.at("Lisa") << endl;
cout << "Hasse --> " << msi["Hasse"] << endl;

auto nisse = msi.find("Nisse");
if(nisse != msi.end()) {
    cout << "Nisse : " << nisse->second << endl;
} else {
    cout << "Nisse not found\n";
}
```

```
Kalle : 1
Kim : 5
Lisa : 2
Lisa --> 2
Hasse --> 0
Nisse not found
```

Avbildningar och mängder

En `std::set` *är i princip en* `std::map` *utan värden*

Operationer på `std::map`

```
insert, emplace, [], at, find, count,
erase, clear, size, empty,
lower_bound, upper_bound, equal_range
```

Operationer på `std::set`

```
insert, find, count,
erase, clear, size, empty,
lower_bound, upper_bound, equal_range
```

Strömmar och filer

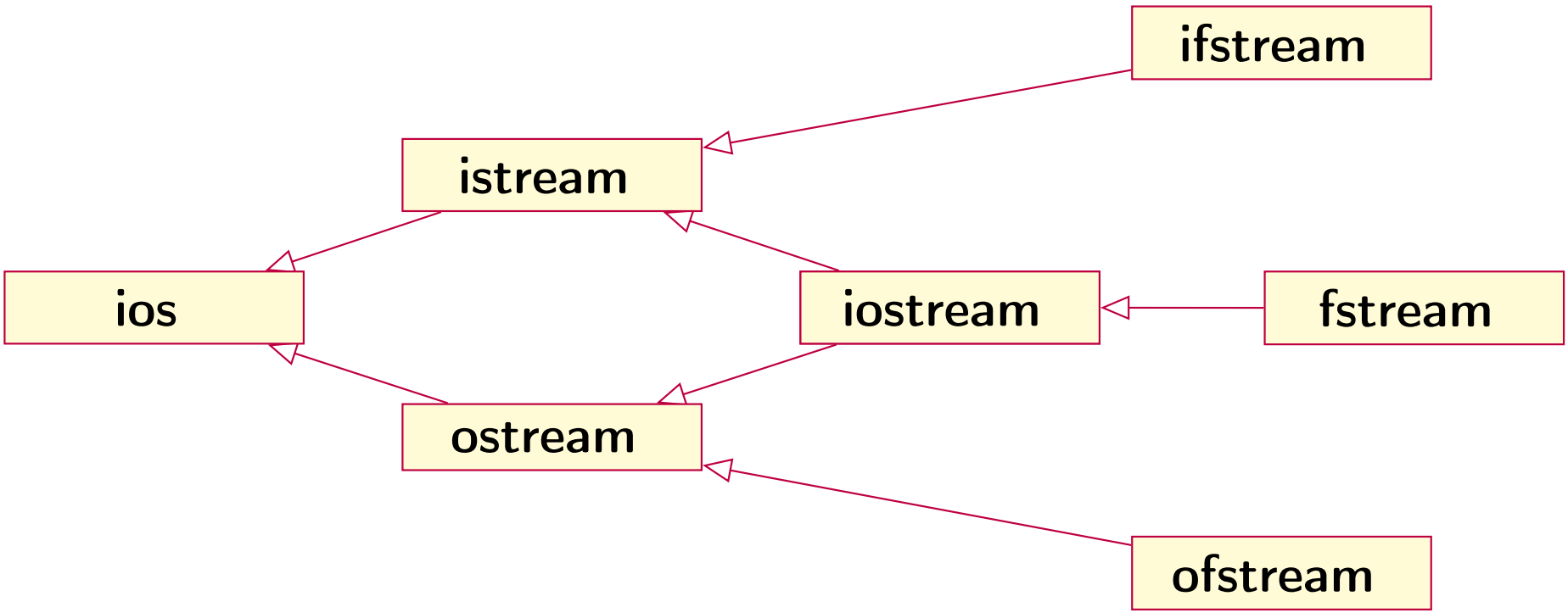
Innehåll

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

Klassen `ios`

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

Klassträd för strömmar



Klassen `ios`

- Fördefinierade strömmar deklarerade i `<iostream>`: `cin`, `cout`, `cerr`, och `clog`
- Klasser deklarerade i inkluderingsfilen: `<fstream>`: `ifstream`, `ofstream`, `fstream`

Läsning av strömmar

Strömmar från klassen `istream` eller från subklass 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 (enligt typen på variablen som ska tilldelas)
  – Görs med operatorn `>>`

Formaterad inmatning

Manipulatorer vid inmatning:

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

Formaterad inmatning

Formaterad inmatning med >> och manipulatorer

```
#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

Oformaterad inmatning med medlemsfunktioner

`int get()`	Läser in och returnerar nästa tecken
`get(char& c)`	Läser in tecken till `c`
`getline(char* s, n, t)`	Läser `n` tecken till `s` med `t` som radseparator
`get(char* s, n, t)`	Som getline men sep. `t` läses ej
`read(char* s, n)`	Läser in `n` st tecken till `s`
`gcount()`	Anger antalet tecken vid senaste inläsn.
`ignore(n, t)`	Hoppar över max `n` st tecken eller till första `t`
`peek()`	Returnerar nästa tecken (som förblir oläst)
`putback(c)`	Lägger tillbaka `c` i strömmen
`unget()`	Lägger tillbaka senast lästa tecken

Klassen `ios`
Statusflaggor

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

Klassen `ios`
Statusflaggor

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()`

cast av en stream `s` till **`bool`** ger `!s.fail()`

Klassen `ios`
Inläsning tecken för tecken

OBS! `istream::eof()` sätts när man *har försökt läsa* `EOF`.

- När `eof` sätts tilldelar inte `get(char& c)` ut-parametern `c`
- men `int get()` returnerar `EOF` (`== -1`)

Utskrift till strömmar

Formaterad utmatning med <<

```
cout << uppercase << "hej svejs" << endl;
```

ger utskriften

```
hej svejs
```

```
cout << scientific << 123456789.0 << endl;
cout << uppercase << scientific << 123456789.0 << endl;
```

ger utskriften

```
1.234568e+08
1.234568E+08
```

Utskrift till strömmar

Manipulatorer vid utmatning:

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

Utskrift till strömmar

Oformaterad utmatning med medlemsfunktioner

`put(char c)`	Skriv ut tecknet `c` till strömmen
`write(const char* s, streamsize n)`	Skriv ut `n` tecken från `s`
`flush()`	Töm utskriftsbufferten

Koppling av filer till strömmar

Öppnande av fil för läsning

```
ifstream infil("infilen.txt");

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

Öppnande av fil för skrivning

```
ofstream utfil("utfilen.txt");

// Alt.
ofstream utfil;
utfil.open("utfilen.txt");
```

Koppling av filer till strömmar

Stängning av fil

```
infil.close();

utfil.close();
```

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

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)){
                f2.put(c);
        }
}
```

Koppling av filer till strömmar

Filflaggor i klassen `ios`

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

Kombination av flaggor med `operator|` (bitvis eller)

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

<stringstream> : strängar som strömmar

```
std::stringstream ss;

ss << "Hello, string!\n";
std::cout << ss.str();

ss.str("Brave new string");

while(ss) {
    std::string s;
    ss >> s;
    std::cout << s << std::endl;
}
```

```
Hello, string!
Brave
new
string
```

Hämta/ändra innehållet i strängen med

- `string stringstream::str() const;`
- `void stringstream::str (const string& s);`

Tips: Använd `stringstream` för att enkelt experimentera med strömmar, eller skriva tester utan konsoll-I/O.