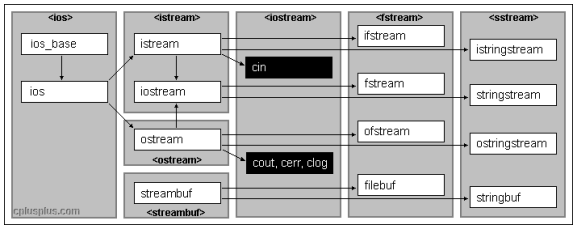
TDP004 Objektorienterad Programmering
Fö 4 Strömmar

Introduktion

- Strömmar används för in- och utmatning.
- En ström är en sekvens av värden, tex teckenström ”byte-ström”.
- Formatterad och oformatterad in/utmatning.
- Samtliga i namespace std.

- Laboration på strömmar.

Översiktbild över streams



Fördefinierade streams

- <iostream> // Generella typer för läsning/skrivning.
  - istream
  - ostream
  - iostream
- <fstream> // Läs/skriv i fil.
  - ifstream
  - ofstream
  - fstream
- <sstream> // Läs/skriv i string.
  - istringstream
  - ostringstream
  - stringstream

In- och utmatning

Exempel:

```
#include <iostream>
using namespace std;
int value;
cout << " Input  a value" << endl;
cin >> value;
cout << "Value was "<< value << endl;
```

Formatterad - oformatterad

- Vid formatterad läsning omvandlas och tolkas texten. White spaces hoppas över (kan styras) och det går att välja vad som läses.
  - char, string, int : ett tecken, ett ord, ett heltal läses in.
  - operatorn >> gör formatterad läsning.
- Vid oformatterad läsning läses data precis som den lagras i strömmen, inget hoppas över eller omvandlas.
  - Alla läsfunktioner utom >>.

```
c = is.get();
getline(cin, is);
c = cin.peek();
```

cin

- För att läsa från konsollen.
- Med "cin >> variabel" lagras värdet i variabel. Det som matas in måste vara av rätt typ, annars avbryts inmatningen och inget lagras.
- En inmatning kan avbrytas med Ctrl-Z (Windows) respektive Ctrl-D (Unix).

cin

Ex:

```
char inputData[100];
cin >> inputData;
```

- Anropas med
  "Preparing lectures takes a long time."
- Vad innehåller inputData?
  Preparing\0

cout

- Skriver ut till konsolllen
- cout << variabel
- Flera anrop kan göras på samma rad:
- Ex:

```
#include <iostream>
int i = 1, j = 5, k = 10;
cout << i << j << k << endl;
```

IO Condition State

- För att kontrollera fel, etc finns olika flaggor och funktioner.
- strm::iostate badbit s.bad()
  strm::iostate failbit s.fail()
  strm::iostate eofbit s.eof()
- s är den stream som man kontrollera. Tillståndet kan sättas med s.setstate(flag) och resetas med s.clear().
- Flag av typen strm::iostate enligt ovan.

Teckenströmmar

- Ostream – ofstream – ostringstream.
- Formatterad skrivning
  - << används vid skrivning.
  - Data tolkas om vid behov, text och tecken skrivs ut som de är. Bool kan skrivas ut som 0/1 eller false/true.
  - failbit sätts om en skrivning misslyckas.
  - Vi kan använda sk manipulatorer vid formatterad skrivning.

Manipulatorer

- Finns i <iomanip>.
- boolalpha/noboolalpha anger hur bool ska skrivas ut.
- left/right vänster- högerjusterat.
- dec/oct/hex anger talbasen vid utskriften.
- setprecision(n) anger antalet decimaler vid utskrift.

  mfl.

Felhantering

```
#include <iostream>
#include <vector>
int main()
{
    vector<int> v;
    int x;
    while(cin>>x)
    {
       v.push_back(x);
    }
    for(unsigned int i = 0; i < v.size(); ++i)
    {
       cout<<v[i]<<endl;
    }
       return 0;
}
```

String stream

- Skriver till/läser från en std::string.
- Ex:

```
#include <sstream>
ostringstream message;
int number = 100;
message << ”Fill with both text and numbers
   like ” <<  number << ”and more text.” << endl;
istringstream source(”Test to copy data”);
string dest;
source >> dest;
```

File stream

- #include <fstream>.
- När man ska läsa eller skriva en fil så öppnas denna med open().
- open kan ta flera argument, bla om existerande fil ska skrivas över, om data ska läggas till sist, om filen enbart ska läsas, etc.
- Detta görs med ”file open modes”.

file open modes

- Används för att avgöra hur en fil ska hanteras:
- in – infil. Öppna filen för läsning.
- out – utfil. Öppna filen för skrivning.
- app – utfil. Öppna filen för skrivning, data läggs till sist i filen. Om filen inte finns skapas den.
- trunc – utfil. Tömmer filen om den finns.
- ate – utfil. Lägger till data på slutet.
- binary – binärfil.

file open modes, forts.

- Flera flaggor kan kombineras mha | (bitvis eller).
- fstream f(”file.bin”, ios:in |
  ios::out |
  ios:binary);
- Filer ska stängs med close().

Filtyper

- Text eller binär.
- En textfil innehåller tecken (char).
  - Skrivs/läses mha cin/cout.
- En binärfil innehåller en följd av bytes (8 bitar).
  - Skrivs/läses mha read/write.
  - Öppnas med ios::binary.

Binärfiler

- Läsning och skrivning görs med read och write.
- read(char* s, streamsize n);
- write(const char* s, streamsize n);
- s är en adress i minnet.
- n anger hur mycket som ska läsas/skrivas.
- storleken på en typ fås genom sizeof<typ>.

Ex. kopiering av binärfil som innehåller int.

```
#include <fstream>
using namespace std;
int main()
{
    ifstream in(”infile.bin”, ios::binary);
    ofstream out(”outfile.bin”, ios::binary);
    int x;
    while(in.read(reinterpret_cast<char*>(&x), sizeof(int) ) )
    {
        out.write(reinterpret_cast<const char*>(&x),
                  sizeof(int) );
    }
    in.close();
    out.close();
    return 0;
}
```

Ex tentauppgift

Du ska göra en del av ett ramverk för att hantera testfall. Gör en funktion enligt följande:

bool WriteTestCaseResult(const unsigned int testCaseNumber,
const std::string testCaseName,
const bool testCaseResult) const;

När funktionen anropas ska en .txt-fil skapas. Denna ska ha namnet testCaseNumber_testCaseName. I filen ska numret, namnet och resultatet skrivas på en rad. Inget mer ska göras med filen. Testfallsresultatet ska skrivas som en bool, och detta ska göras med existerande funktionalitet, dvs du ska **inte** skriva en egen if-sats som skriver strängen "false" resp "true" i filen. Ingenting mer ska skrivas i filen. Om funktionen misslyckas med att skapa filen ska funktionen returnera false, annars ska funktionen returnera true. Om det redan finns en fil med samma namn ska denna skrivas över.

Exempel:

WriteTestCaseResult(10, testsearch, true)

-> I filen 10_testsearch.txt

10 testsearch true

Ex tentauppgift, lösningsförslag

```
bool Uppg2::WriteTestCaseResult(const unsigned int testCaseNumber,
                                const std::string testCaseName,
                                const bool testCaseResult) const
{
    std::stringstream ss;
    ss << testCaseNumber << "_" << testCaseName << ".txt";
    std::string fileName = ss.str();
    std::ofstream saveFile(fileName.c_str());
    if(NULL == saveFile)
    {
        return false;
    }
    saveFile << testCaseNumber
        << " "
        << testCaseName
        << " "
        << std::boolalpha << testCaseResult
        << '\n';
    saveFile.close();
    return true;
}
```

Användande av stream

- Används för in- utmatning.
- Debuggning – det är inte specificerat i vilken ordning olika objekt initieras, cout kan initieras efter alla dina objekt. *Om cout anropas i en konstruktor är resultatet odefinierat.*

Sammanfattning

- Olika sorters strömmar:
  - filestreams
  - stringstreams
  - Iostreams
- Möjligt att specificera om en ström ska användas för in/utmatning eller båda.
- Manipulatorer.
- File open modes.