TDIU01 - Programmering i C++, grundkurs

Strömmar, externa filer och kommandoradsargument

Eric Elfving
Institutionen för datavetenskap

7 oktober 2015

LINKÖPINGS UNIVERSITET

Översikt

- Strömmar
- Kommandoradsargument
- Jämförelseoperatorer

Strömmar

- En ström är en kanal där tecken kan skickas från en enhet till en annan.
- Vi har redan använt två strömmar, standard inmatningsström (cin) där tecken skickas från tangentbordet till vårt program och standard utmatningsström (cout) där tecken skickas från vårt program till skrämen.
- Båda dessa strömmar är buffrade, dvs operativsystemet mellanlagrar allt som skrivs till dessa strömmar i en intern buffer som tillhör vårt program.

cin och cout är buffrade...

cerr

- Det finns en annan vanlig ström i C++ (och terminalen), cerr
- `cerr`, eller felströmmen är en obuffrad variant av cout.
- Om man skriver felmeddelanden till cerr istället för cout garanteras utskriften.
- Kodexempel 1...

cerr

- Felmeddelanden kan även hanteras på ett speciellt sätt av terminalen.
- Följande kommando skriver all vanlig utskrift till filen `utdata.log` och felmeddelanden till filen `error.log`
  - BASH: `a.out > utdata.log 2> error.log`

Felflaggor

- Om något går fel vid inläsning eller utskrift sätts en felflagga för strömmen.
- Följande felflaggor finns för alla strömmar:

`Felflagga`	Beskrivning	Funktioner för att kontrollera status			
		`good()`	`eof()`	`fail()`	`bad()`
`goodbit`	Inget fel	`true`	`false`	`false`	`false`
`eofbit`	Filslut (ctrl-d)	`false`	`true`	`false`	`false`
`failbit`	Logiskt fel	`false`	`false`	`true`	`false`
`badbit`	Annat fel	`false`	`false`	`true`	`true`

- Med funktionen `clear()` kan man ta bort alla felflaggor.

Filströmmar

- En filström är en ström som kan kopplas till en extern textfil på hårddisken för att låta oss läsa eller skriva från denna fil istället för att kommunicera med användaren.
- Det finns tre typer av filströmmar:

fstream	Du bestämmer hur filen ska hanteras
ifstream	Filen öppnas för läsning, du får en inläsningsström
ofstream	Filen öppnas för skrivning, du får en utmatningsström

Filströmmar

- Filer kan öppnas på två sätt, vid initiering och med funktionen open:

```
#include <fstream> // för att komma åt filströmmar
#include <iostream>
using namespace std;

int main()
{
  ifstream in_file {"indata.txt"};
  ofstream out_file;
  cout << "Vart ska resultatet hamna? Filnamn: ";
  string filename;
  cin >> filename;
  out_file.open(filename);
  // copy contents of indata.txt to the given file:
  char c;
  while (in_file.get(c))
    out_file << c;
}
```

Filströmmar

Filmoder

- En filmod beskriver hur filen ska öppnas, nedan visas de vanligaste:

`Filmod`	Betydelse
`ios::in`	Inläsning, börjar från början av filen
`ios::out`	Utskrift, innehåll trunkeras och start i början av filen
`ios::app`	Utskrift, skriver till slutet av filen

- Om man har en `fstream` måste man kombinera `out` och `app` för att skriva till slutet: `"ios::out | ios::app"`
- Man väljer filmod när man öppnar filen:
  `ofstream ofs{"filnamn", ios::app};`

Filströmmar

- Precis som med pekare är det viktigt att återlämna de resurser vi använder
- För filer betyder det att vi bör stänga öppnade filer när vi är klara med dem
- Det görs med funktionen `close`

Filströmmar


```
int main()
{
  ifstream ifs {"indata.txt"};
  if (!ifs) //kallar på ifs.fail()
  {
    cerr << "Couldn't open input file";
    return 1;
  }
  ofstream ofs {"output.txt"};
  if (!ofs)
  {
    cerr << "Couldn't open output file!";
    ifs.close();
    return 2;
  }
  // do some processing ...
  ifs.close();
  ofs.close();
}
```

Strängströmmar

- En strängström liknar en filström men är kopplad till en sträng istället för en fil.
- Används ofta för att konvertera mellan olika datatyper:

```
#include <sstream>
using namespace std;
int string_to_int(const string & str)
{
  stringstream sst;
  sst << str;
  int d;
  sst >> d;
  return d;
}
string to_string(int d)
{
  stringstream sst;
  sst << d;
  return sst.str(); //Ta fram den bakomliggande strängen
}
```

Strängströmmar

- Det finns tre varianter, `stringstream`, `istringstream` och `ostringstream`.
- Med `stringstream` kan man både läsa och skriva. istringstream går bara att läsa ifrån och ostringstream går bara att skriva till.

```
#include <sstream>
using namespace std;
int string_to_int(const string & str)
{
  istringstream iss {str}; //initiera med strängen str
  int d;
  iss >> d;
  return d;
}
string int_to_string(int d)
{
  ostringstream oss;
  oss << d;
  return oss.str();
}
```

Inläsningsströmmar

- När vi läser från en inmatningsström tolkas tecknen som finns i strömmen enligt datatypen på variabeln där vi vill lagra resultatet av inläsningen.
- Om något går fel vid den tolkningen sätts failbit. Om något går fel vid inläsningen eller om något annat är fel sätts badbit.
- Man kan använda namnet `istream` som ett samlingsnamn på alla inmatningsströmmar (cin, ifstream, istringstream).

Utmatningsströmmar

- När vi skriver till en utmatningsström konverteras värdet på det vi skriver ut till en sekvens av tecken.
- Man kan använda namnet `ostream` som ett samlingsnamn på alla utmatningsströmmar (cout, cerr, ofstream, ostringstream).

Exempel


```
struct Book
{
  string title;
  string author;
  int pages;
};

void print(const Book & b)
{
  cout << b.author << ": " << b.title
       << " (" << b.pages << " pages)" << endl;
}
int main()
{
  Book b {"C++ direkt", "Skansholm", 657};
  print(b);
}
```

Exempel


```
struct Book
{
  string title;
  string author;
  int pages;
};

void print(const Book & b)
{
  cout << b.author << ": " << b.title
       << " (" << b.pages << " pages)" << endl;
}
int main()
{
  Book b {"C++ direkt", "Skansholm", 657};
  print(b);
}
```

Exempel


```
void print(const Book & b)
{
  cout << b.author << ": " << b.title
       << " (" << b.pages << " pages)" << endl;
}
void print(const Book & b, ofstream & f)
{
  f << b.author << ": " << b.title
    << " (" << b.pages << " pages)" << endl;
}
int main()
{
  Book b {"C++ direkt", "Skansholm", 657};
  ofstream output{"filnamn.txt"};
  print(b, output);
}
```

Exempel


```
void print(const Book & b)
{
  cout << b.author << ": " << b.title
       << " (" << b.pages << " pages)" << endl;
}

void print(const Book & b, ofstream & f)
{
  f << b.author << ": " << b.title
    << " (" << b.pages << " pages)" << endl;
}
int main()
{
  Book b {"C++ direkt", "Skansholm", 657};
  ofstream output{"filnamn.txt"};
  print(b, output);
}
```

Exempel


```
void print(const Book & b)
{
  cout << b.author << ": " << b.title
       << " (" << b.pages << " pages)" << endl;
}
```

```
void print(const Book & b, ostream & f)
{
  f << b.author << ": " << b.title
    << " (" << b.pages << " pages)" << endl;
}
int main()
{
  Book b {"C++ direkt", "Skansholm", 657};
  ofstream output{"filnamn.txt"};
  print(b, output);
}
```

Exempel


```
   void print(const Book & b, ostream & f = std::cout)
{
  f << b.author << ": " << b.title
    << " (" << b.pages << " pages)" << endl;
}
int main()
{
  Book b {"C++ direkt", "Skansholm", 657};
  ofstream output{"filnamn.txt"};
  print(b,output);
  print(b); // prints to standard output
}
```

Kommandoradsargument

- Terminalen har ett inbyggt sätt att skicka data till vårt program redan vid programstart kallat kommandoradsargument.
- Man startar sitt program med dess namn följt av argumenten som ska skickas.
- Vi har använt kommandoradsargument ganska mycket redan: `g++ -o program fil.cc` betyder ``starta programmet `g++` och skicka argumenten `-o, program` och `fil.cc`.

Kommandoradsargument

- De flesta språk har mekanismer för att komma åt dessa argument.
- C++ använder sig av mekanismer ärvda från C.
- För att komma åt argumenten ändrar man sitt funktionshuvud för main till `int main(int argc, char* argv[])`.
- `argc` (argument count) kommer sättas till antal argument (inklusive programnamnet).
- `argv` (argument vector) blir ett fält av c-strängar innehåller argumenten...

Fält

- Ett fält är en sekvensiell datastruktur liknande `vector` men har en fast storlek som den inte känner till. I grunden är det av en pekare till första elementet.
- Med deklarationen `int arr[4]` får vi ett fält som kan lagra fyra heltal.
- Namnet `arr` fungerar som en pekare till första elementet i fältet. `arr[i]` tolkas som `*(arr + i)` (eller hoppa `i` heltal framåt i minnet från position `arr`).

c-strängar

- En c-sträng är föregångaren till `string` och är i grunden ett fält av tecken. Eftersom den inte känner till sin egen storlek avslutas fältet med tecknet `'\0'` (`char{0}`) av konvention.
- Eftersom c-strängar i grunden är pekare fungerar INTE de vanliga jämförelseoperatorerna på dessa. Man måste använda speciella funktioner istället (finns i `<cstring>`).

Kommandoradsargument

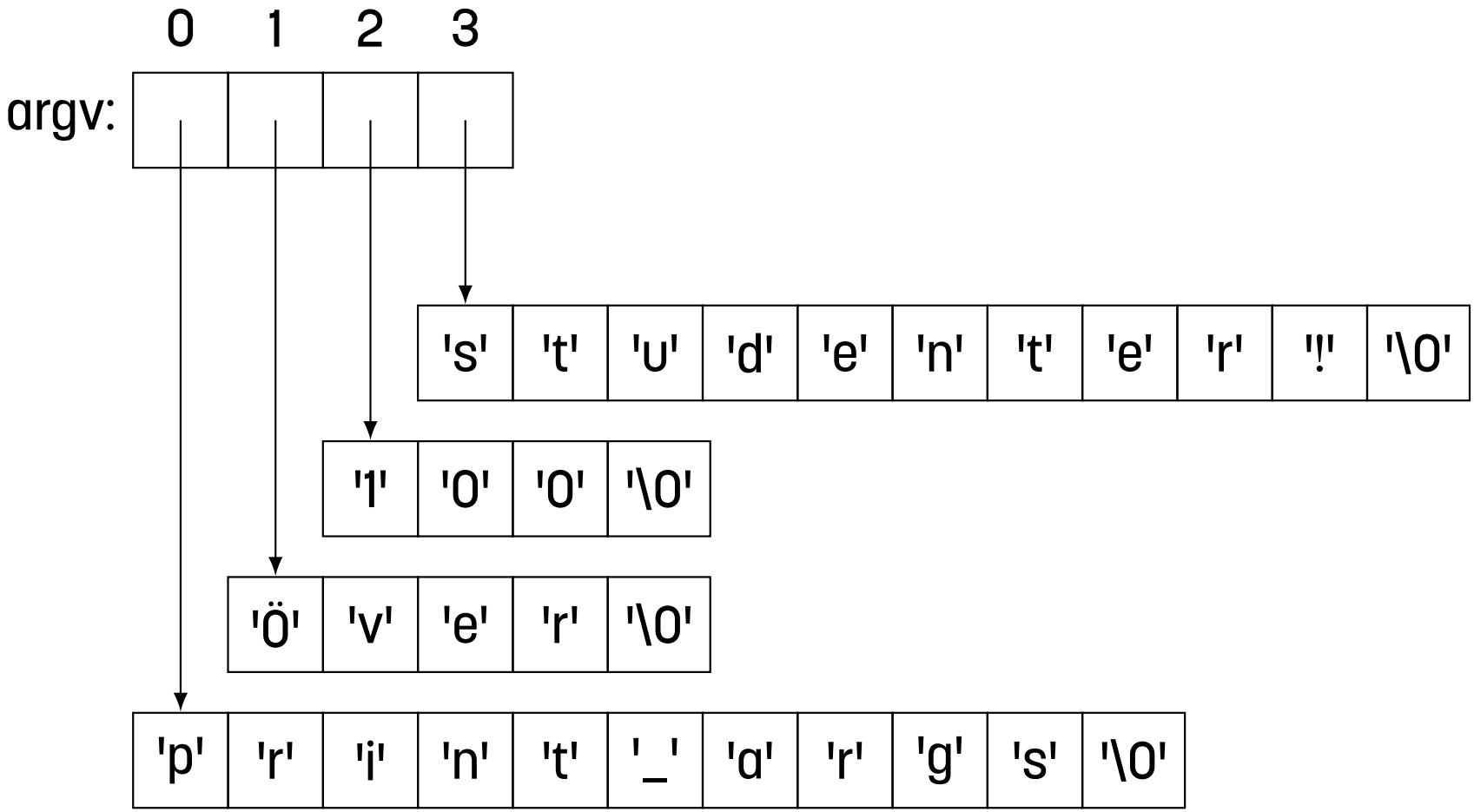
- Antag att vi har följande program kompilerat till namnet `print_args`:

```
int main(int argc, char* argv[])
{
  for (int arg=1; arg <argc; ++arg)
  {
    cout << "Argument #" << arg << ": "
         << argv[arg] << endl;
  }
}
```

- Programkörning:

```
zaza10$ print_args Över 100 studenter!
Argument #1: Över
Argument #2: 100
Argument #3: studenter!
```

Kommandoradsargument



Jämförelseoperatorer

- Vi kan skapa funktioner för att jämföra egna posttyper.
- Antingen skapar vi en vanlig funktion eller så överlagrar vi en (binär) jämförelseoperator.
- Vi kan överlagra alla inbyggda operatorer, dock inte skapa egna.
- Får användas på labben, men inget krav. Mer övning kommer i senare kurser.

Jämförelseoperatorer


```
struct Book
{
  string title;
  string author;
  int pages;
};

bool operator<(const Book & lhs, const Book & rhs)
{
  return lhs.author < rhs.author ||
       (lhs.author == rhs.author && lhs.title < rhs.title);
}
```

Jämförelseoperatorer

Följande uttryck:

```
if ( b1 < b2 )
  ...
```

Byts ut mot detta anrop av kompilatorn:

```
if ( operator<(b1,b2) )
  ...
```

www.liu.se

LINKÖPINGS UNIVERSITET