



THE C++ PROGRAMMING LANGUAGE

CPP-06 – C++ Standard Library
CPVR Vertiefungsmodul BTI-7281
Urs Künzler (urs.kuenzler@bfh.ch)

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CPP-06		Table of Contents – Course Introduction	
06.01	Allgemeine Konzepte		3
06.02	C++ Strings (Standard Library)		7
06.03	I/O Streams		11
06.04	File I/O Streams		22
06.05	C++ String I/O Streams		29


CPP-09	2
--------	---

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

06.01

Allgemeine Konzepte



- Pre-ANSI Standard Implementationen von C++ Compiler beinhalteten nur die von ANSI C übernommene C Library
- Nachteil: unterschiedliche Implementation von allgemeinen Hilfsklassen durch verschiedene C++ Compiler Hersteller (z.B. String-Klassen)
- Die C++ Standardbibliothek bildet einen integralen Bestandteil des ANSI Standards, wodurch jeder C++ Standard konforme Compiler diese Library unterstützen muss. 

CPP-09

3

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Aufgabenbereiche der Standard Library



- Die Standardbibliothek ist in verschiedene Aufgabenbereiche unterteilt:
 - Language Support: dynamische Speicherverwaltung, RTTI
 - Diagnostics: Exceptions, Assertions, Fehlercodes
 - Strings: String-Verarbeitung
 - I/O-Streams: Ein- und Ausgabe-Streams, Formatierung, Manipulatoren
 - Numerics: komplexe Zahlen und numerische Arrays
 - General Utilities: allgemeine verwendbare Klassen und Funktionen
 - Localization: Lokalisierte Datendarstellung (Datum, Zeit, Währung, etc.)
 - Containers (STL): Objektverwaltung (vector, deque, list, set, map)
 - Iterators (STL): Iteratoren für Container
 - Algorithms (STL): Algorithmen für Container (Suchen, Sortieren, etc.)
- Die C Library ist ebenfalls Bestandteil der C++ Standardbibliothek

CPP-09

4

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Namensbereich (Namespace)



- Um Namenskonflikte zwischen verschiedenen Modulen oder Libs zu vermeiden, können in C++ Namensbereiche verwendet werden.
- Ein Namensbereich unterteilt den globalen Namensraum und definiert einen Geltungsbereich (Scope) für einen Identifier.
 - Beispiel:

```
namespace A{void f(void);};
namespace B{void f(void);};
```
- Die Qualifizierung des Namensbereichs erfolgt mit Hilfe des Scope-Operators.
 - Beispiel:

```
A::f(); B::f();
```
- Pro File können mehrere Namensbereiche definiert werden und diese können auch verschachtelt sein.
- Mit dem Keyword `using` wird der (default) Namespace angegeben.
 - Beispiel:

```
using namespace A;
```

CPP-09

5

Demo: CPP-06-D.01 - Namespace

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Standard Library Namespace "std"



- Alle C++ Standard Library Funktionen (inkl. C Runtime Library) sind im Namespace `std` deklariert.
- Header Files der C++ Standard Library werden ohne Filename Extension eingebunden.
 - Beispiel:

```
#include <iostream>
```
- Dateinamen von C Header Files werden mit einem vorangestellten "c" übernommen.
 - Beispiel:

```
#include <cstdlib> // anstelle #include<stdlib.h>
```
- Aus Kompatibilitätsgründen sind die "normalen" C Header Files ebenfalls vorhanden, allerdings sind diese dann nicht im Standard Namespace (`std`) definiert.

CPP-09

6

Demo: <cstdlib> und <stdlib.h>

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

06.02

C++ Strings (Standard Library)



- Die Standard Library definiert im Header File `<string>` mehrere String-Klassen:
 - `basic_string` ist die String-Template-Klasse, welche die ganze String-Funktionalität unabhängig von einem speziellen Zeichentyp implementiert.
 - `string` ist die konkret instanzierte Template-String-Klasse für Strings mit Zeichen vom Typ `char`. (Deklaration: `typedef basic_string<char> string;`)
 - `wstring` ist eine weitere konkret instanzierte Template-String-Klasse für Strings mit Zeichen vom Typ `wchar_t`. Diese Strings dienen der Internationalisierung. (Deklaration: `typedef basic_string<wchar_t> wstring;`)
- Die so definierten String-Klassen können wie vordefinierte Datentypen verwendet werden, wobei die Buffer-Grösse falls notwendig automatisch erhöht wird.



CPP-09

7

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

Standard Library Strings II



- Konstruktion, Kopieren und Löschen von Strings:
 - `string s0` erzeugt einen Leer-String
 - `string s1(s0)` erzeugt einen String als Kopie eines andern Strings
 - `string s2(10, ' ')` erzeugt einen String mit 10 Leerzeichen
 - `string s3("text")` erzeugt einen C++ String aus einer C-String Konstanten
 - ...
 - `~string()` Destruktor, löscht den String und gibt den Speicher frei
- Nicht modifizierende String-Funktionen:
 - `size()` liefert die Anzahl Zeichen im String
 - `length()` liefert die Anzahl Zeichen im String (äquivalent zu `size()`)
 - `empty()` testet ob der String leer ist
 - `max_size()` liefert die max. Anzahl Zeichen die der String enthalten kann
 - `reserve(anz)` reserviert String Speicherplatz für mindestens Anzahl Zeichen
 - `capacity()` liefert die Anzahl Zeichen, die ohne Reallozierung vom String aufgenommen werden können



CPP-09

8

Demo: CPP-06-D.02 - C++ Strings

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

Standard Library Strings III



- **Modifizierende String-Funktionen:**

- `operator+=()` anhängen eines Strings am Ende eines bestehenden Strings
- `operator+()` verketten von Strings
- `operator=()` zuweisen von Strings
- `assign()` zuweisen von Strings (versch. Varianten)
- `insert()` einfügen von Strings (versch. Varianten)
- `replace()` ersetzen von Strings (versch. Varianten)
- `erase()` löschen von Zeichen (versch. Varianten)
- `resize(anz, ' ')` setzen der Länge eines Strings (versch. Varianten)

...

- **String-Funktionen zum suchen:**

- `find()` sucht ersten Teil-String (versch. Varianten)
- `rfind()` sucht letzten Teil-String (versch. Varianten)
- `find_first_of()` sucht erstes Zeichen einer Menge (versch. Varianten)
- `find_last_of()` sucht letztes Zeichen einer Menge (versch. Varianten)

...

CPP-09

9

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Standard Library Strings IV



- **String Vergleichsoperatoren:**

- `==, !=, <, >, <=, >=` Standard Vergleichsoperatoren
- `compare()` testet zwei Strings auf Gleichheit (versch. Varianten)

- **String-Funktionen für den direkten Zeichenzugriff:**

- `at(index)` Zeichenzugriff an der Stelle index (prüft Indexgrenzen)
- `operator[index]` Zeichenzugriff an der Stelle index (prüft Indexgrenzen nicht)
- `c_str()` konvertiert ein String-Objekt in einen C-String (Null terminiertes Char Array)

- **String-Funktionen für Ein- und Ausgabe:**

- `operator<<()` ausgeben eines Strings auf dem Output-Stream
- `operator>>()` einlesen einer Zeichenfolge vom Input-Stream bis zum ersten Trennzeichen (Whitespace).
- `getline()` einlesen einer Zeichenfolge vom Input-Stream bis zum ersten Newline-Zeichen (versch. Varianten)

CPP-09

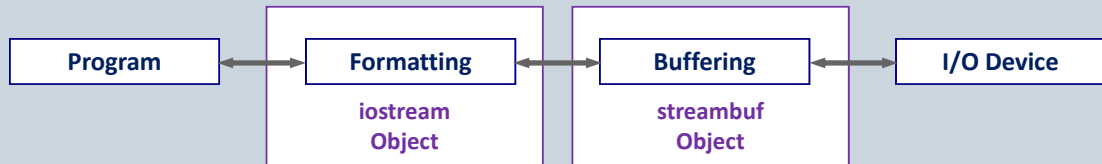
10

Demo: CPP-06-D.02 - String Bsp. Wort Extraktion

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

06.03

I/O Streams



- Die Ein- und Ausgabe erfolgt in C++ mittels sogenannter Streams. Dies gilt nicht nur für Tastatur/Bildschirm sondern auch für Files.
- Ein Stream kann als “Datenstrom” verstanden werden von dem Daten gelesen oder hinein geschrieben werden können.
- Streams sind als Klassen implementiert, wobei man zwischen Input Streams (istream) für die Eingabe und Output Streams (ostream) für die Ausgabe unterscheidet.

CPP-09

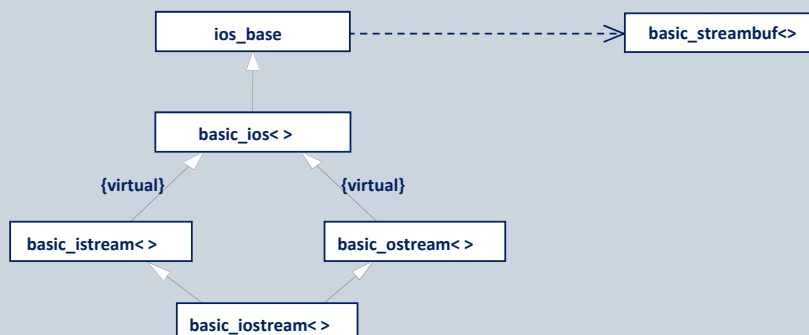
11

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

I/O Streams II



- Die Klassen zur Implementation von Streams sind in folgender Klassenhierarchie organisiert:



CPP-09

12

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

I/O Streams III



- **ios_base**

- Basisklasse welche die allgemeinen Eigenschaften aller Stream-Klassen unabhängig von einem speziellen Zeichentyp definiert (z.B. Status- und Format-Flags).

- **basic_ios<>**

- Diese von der Basisklasse abgeleitete Template-Klasse definiert die allgemeinen Eigenschaften der Stream-Klassen die vom speziellen Zeichentyp abhängig sind (z.B. Definition des zum Zeichentyp gehörenden Datenpuffers).
- **ios** - ist die konkret instanzierte Template-Klasse für den Zeichentyp `char`.
(Deklaration: `typedef basic_ios<char> ios;`)

CPP-09

13

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

I/O Streams IV



- **basic_streambuf<>**

- Template-Klasse zur Implementation der gepufferten Lese- und Schreiboperationen auf einen spezifischen "Datenkanal".
- **streambuf** - ist die konkret instanzierte Template-Klasse für den Zeichentyp `char`.
(Deklaration: `typedef basic_streambuf<char> streambuf;`)

- **basic_istream<>** bzw. **basic_ostream<>**

- Template-Klassen für die Implementation der Input- bzw. Output- spezifischen Stream-Funktionen.
- **istream** bzw. **ostream** - sind die konkret instanziierten Template-Klassen für den Zeichentyp `char`. (Deklaration: `typedef basic_istream<char> istream;` bzw. `typedef basic_ostream<char> ostream;`)

- **basic_iostream<>**

- **iostream** ist die `char` Klassen-Instanz für die kombinierte Stream Ein- und Ausgabe.
(Deklaration: `typedef basic_iostream<char> iostream;`)

CPP-09

14

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

I/O Streams V



- Für die Standard Ein- und Ausgabe Streams gibt es die folgenden vordefinierten globalen Objekte im `std` Namespace:
 - `cin` - Stream-Objekt für den Standard-Eingabekanal, entspricht der C-Variable `stdin`. (Definition: `istream cin;`)
 - `cout` - Stream-Objekt für den Standard-Ausgabekanal, entspricht der C-Variable `stdout`. (Definition: `ostream cout;`)
 - `cerr` - Stream-Objekt des Standard-Ausgabekanals für Fehler, entspricht `stderr`. Die Ausgaben auf `cerr` werden nicht gepuffert. (Definition: `ostream cerr;`)
 - `clog` - Analog `cerr`, die Ausgaben erfolgen jedoch gepuffert. In C ist dafür keine entsprechende Variable definiert. (Definition: `ostream clog;`)
- Durch include des Header Files `<iostream>` werden alle notwendigen Stream-Klassen und die vordefinierten Stream-Objekte in eigene Programme eingebunden.

CPP-09

15

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

I/O Streams: Funktionen



- **Standard Stream-Operatoren:**
Die Operatoren `operator>>()` und `operator<<()` sind für alle vordefinierten Datentypen überladen und bilden die Standardoperationen für die Stream Ein- und Ausgabe. Für Ein- und Ausgabe eigener Datentypen werden mit Vorteil ebenfalls diese Operatoren überladen.
- **Stream-Funktionen für die Eingabe:**
 - `get()` liefert das nächste gelesene Zeichen oder EOF zurück (versch. Varianten).
 - `getline()` liest eine ganze Zeile bis zum EOF oder '\n' Zeichen.
 - `read()` liest eine maximale Anzahl Zeichen in einen C-String.
 - `peek()` liefert nächstes Zeichen aus dem Stream oder EOF, ohne es auszulesen. (Beim nächsten Lesen wäre das Zeichen also immer noch vorhanden.)
 - `gcount()` liefert die Anzahl gelesener Zeichen des letzten Lesebefehls.
 - ...
- **Stream-Funktionen für die Ausgabe:**
 - `put()` gibt das übergebene Zeichen als nächstes Zeichen aus.
 - `write()` schreibt eine spezifizierte Anzahl Zeichen eines C-Strings.
 - `flush()` leert den Ausgabepuffer, indem alle Zeichen ausgegeben werden.

CPP-09

16

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

I/O Streams: Stream State



- Zum Erkennen einer Fehlersituation verfügen Streams über einen Zustand der durch **Fehlerzustands-Flags** beschrieben wird:
 - `ios::goodbit` alles in Ordnung (kein Fehler-Bit gesetzt)
 - `ios::eofbit` end of file - das Ende der Daten ist erreicht
 - `ios::failbit` Fehler - der letzte Vorgang wurde nicht korrekt abgeschlossen (z.B. ungültiges Zeichen für int Typ eingelesen)
 - `ios::badbit` fataler Fehler - Zustand des Streams ist nicht definiert (z.B. Disk voll)
- Ist eines der drei Fehler-Flags gesetzt, wird der entsprechende Stream für die weitere Ein- oder Ausgabe gesperrt. Zurücksetzen erfolgt mit der Funktion `clear()`.
- Funktionen zum Setzen und Abfragen des Stream-Zustandes:
 - `good()` true wenn alles in Ordnung ist (`ios::goodbit` gesetzt)
 - `eof()` true bei end of file (`ios::eofbit` gesetzt)
 - `fail()` true bei Fehler (`ios::failbit` oder `ios::badbit` gesetzt)
 - `bad()` true bei fatalem Fehler (`ios::badbit` gesetzt)
 - `rdstate()` liefert die Kombination der gesetzten Zustands-Flags
 - `clear()` löscht alle gesetzten *Fehlerzustands-Flags*
 - `exceptions()` aktiviert Exception-Mechanismus bei Änderungen des Zustands

CPP-09

17

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

I/O Streams: Manipulatoren



- Manipulatoren sind spezielle Stream Funktionen, deren Ausgabe eine Manipulation am Stream durchführt. Damit kann z.B. das Ausgabeformat entsprechend eingestellt werden.
- Im Header File `<iostream>` sind die folgenden häufig verwendeten Manipulatoren definiert:
 - `endl` Newline ausgeben und Ausgabepuffer leeren
 - `flush` Ausgabepuffer leeren
 - `ws` Whitespace bei der Eingabe ignorieren
- Eigene Stream-Manipulatoren können ebenfalls definiert werden.

CPP-09

18

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

I/O Streams: Formatdefinitionen



- Zur Definition des Ein- bzw. Ausgabeformates werden in der Stream- Klasse `ios_base` verschiedene Bitfelder für Format-Flags definiert.
- Zum Setzen oder Abfragen der Stream Format-Flags können prinzipiell zwei Möglichkeiten verwendet werden:
 - Direkter Format-Flag Zugriff mit den Funktionen `setf()`, `unsetf()` und `flags()`.
 - Zugriff mittels den allg. Manipulatoren `setiosflags()` bzw. `resetiosflags()` oder mit speziell definierten Manipulatoren für bestimmte Format-Flags. Diese Format-Flag Manipulatoren werden durch das Header File `<iomanip>` eingebunden.

- Beispiele (alle Varianten sind äquivalent):

```
// Setzen der Flags "ios::showbase" und "ios::hex"
cout.setf(ios::showbase | ios::hex); cout << 255 << endl;
cout << setiosflags(ios::showbase | ios::hex) << 255 << endl;
cout << showbase << hex << 255 << endl;

cout.flags(0); // alle Format-Flags zurücksetzen
```

CPP-09

19

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

I/O Streams: Formatdefinitionen II



- **Definition der Feldbreite, Füllzeichen und Ausrichtung:**
 - `width()` Funktion zum Setzen der Feldbreite
 - `fill()` Funktion zum Setzen des Füllzeichens (Default: ' ')
 - `ios::left` Format-Flag für linksbündig
 - `ios::right` Format-Flag für rechtsbündig
 - `ios::internal` Format-Flag für Vorzeichen linksbündig und Wert rechtsbündig
- **Vorzeichen und Grossbuchstaben numerischer Werte:**
 - `ios::showpos` Format-Flag um positives Vorzeichen auszugeben
 - `ios::uppercase` Format-Flag für grosse Buchstaben in numerischen Werten
- **Definition des Zahlensystems:**
 - `ios::oct` Format-Flag für octale Ein- und Ausgabe
 - `ios::dec` Format-Flag für dezimale Ein- und Ausgabe (default)
 - `ios::hex` Format-Flag für hexadezimale Ein- und Ausgabe
 - `ios::showbase` Format-Flag um das Zahlensystem zu kennzeichnen

CPP-09

20

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

I/O Streams: Formatdefinitionen III



- **Definition der Gleitkommanotation:**

- `ios::fixed` Format-Flag für Dezimaldarstellung von Gleitkommazahlen
- `ios::scientific` Format-Flag für Exponentialdarstellung von Gleitkommazahlen
- `precision()` Funktion zum Setzen oder Abfragen der Genauigkeit von Gleitkommazahlen, wobei die Werte entsprechend gerundet werden.

- **Allgemeine Format-Flags:**

- `ios::skipws` Format-Flag um führende Whitespaces (Bsp. Blancs) zu überlesen
- `ios::unitbuf` Format-Flag um den Ausgabepuffer nach jeder Ausgabeoperationen zu leeren

CPP-09

21

Demo: CPP-06-D.03/D.04 - ASCII Table Output (Manipulators/Set)

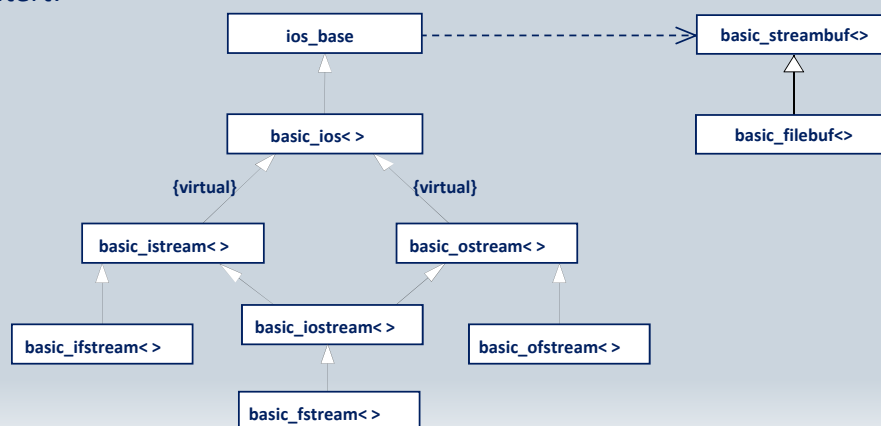
 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

06.04

File I/O Streams



- Streams können auch verwendet werden um Files zu Lesen und zu Schreiben. Zu diesem Zweck wird die Stream-Klassenhierarchie um spezifische File-Stream-Klassen erweitert:



CPP-09

22

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

File I/O Streams II



- **basic_ifstream<>**
 - Abgeleitete Template-Klasse für Input File Stream spezifische Funktionen.
 - **ifstream** - ist die konkret instanzierte Template-Klasse für den Zeichentyp char.
(Deklaration: `typedef basic_ifstream<char> ifstream;`)
- **basic_ofstream<>**
 - Abgeleitete Template-Klasse für Output File Stream spezifische Funktionen.
 - **ofstream** - ist die konkret instanzierte Template-Klasse für den Zeichentyp char.
(Deklaration: `typedef basic_ofstream<char> ofstream;`)
- **basic_fstream<>**
 - **fstream** - char Klassen-Instanz für die kombinierte Ein- und Ausgabe auf File-Streams.
(Deklaration: `typedef basic_fstream<char> fstream;`)
- **basic_filebuf<>**
 - Abgeleitete Template-Klasse für File Buffer spezifische Funktionen.
 - **filebuf** - ist die konkret instanzierte Template-Klasse für den Zeichentyp char.
(Deklaration: `typedef basic_filebuf<char> filebuf;`)

CPP-09

23

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

File I/O Streams III



- Durch Ableitung der File-Stream-Klassen von “normalen” Streams können die Funktionen und Operatoren für die eigentliche Ein- und Ausgabe von Streams (<<, >>, get(), getline(), put() etc.) auch zum Lesen und Schreiben von Files verwendet werden.
- Das Gleiche gilt für die von den “normalen” Stream-Klassen bekannten Fehlerzustands-Flags und Formatdefinitionen, welche sämtliche auch für die File Ein- und Ausgabe verwendbar sind.
- Durch include des Header Files `<fstream>` werden alle notwendigen File-Stream-Klassen in eigene Programme eingebunden.
- Leider fehlt der Zugriff auf allgemeine File Informationen wie z.B. File-Grösse oder File-Attribute. Für diese Informationen müssen (plattformabhängige) System-Calls verwendet werden.

CPP-09

24

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

File I/O Streams: Files öffnen



- Ein Vorteil bei der Verwendung von File Streams besteht darin, dass Files nicht explizit geöffnet werden müssen, sondern dies im Konstruktor des File Stream Objektes “automatisch” erfolgt.
 - Beispiel: `ofstream txtfile("myfile.txt");`
- Zur genauen Spezifikation einer File-Operation sind die folgenden File-Flags definiert:
 - `ios::in` Lesen (Default bei ifstream)
 - `ios::out` Schreiben (Default bei ofstream)
 - `ios::app` Anhängen (append)
 - `ios::ate` am File Ende positionieren (at end)
 - `ios::trunc` alten File Inhalt löschen (truncate)
 - `ios::binary` File im binären Modus öffnen
 - Beispiel: `ofstream txtfile("myfile.txt", ios::app | ios::binary);`
- Nur fstream Objekte können durch Angabe der entsprechenden Flags zum Lesen und Schreiben geöffnet werden!

CPP-09

25

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

File I/O Streams: Files schliessen



- Analog zum Öffnen erfolgt das Schliessen eines Files im Destruktor des File Stream Objektes. Ein File muss deshalb nicht explizit geschlossen werden, sondern dies wird “automatisch” beim Löschen des Objektes durchgeführt.
- Beispiel:


```
// statisches Erzeugen eines File Streams
ifstream infile("input.txt");

// dynamisches Erzeugen eines File Streams
ofstream* p_outfile = new ofstream("output.txt");

...
// explizites Löschen des dynamisch erzeugten File Streams
// schliesst das dazugehörige File
delete p_outfile;

// Beim Verlassen des Scoops wird der statisch erzeugte File Stream // automatisch
gelöscht und das dazugehörige File geschlossen.
```

CPP-09

26

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

File I/O Streams: Positionierung im File



- Für die freie Positionierung des aktuellen File-Cursors sind folgende Funktionen und File-Flags definiert:

▪ <code>seekg()</code>	Funktion zum Setzen einer Lese-Position (seek get)
▪ <code>tellg()</code>	Funktion welche die aktuelle Lese-Position liefert (tell get)
▪ <code>seekp()</code>	Funktion zum Setzen einer Schreib-Position (seek put)
▪ <code>tellp()</code>	Funktion welche die aktuelle Schreib-Position liefert (tell put)
▪ <code>ios::beg</code>	File-Flag für eine Position relativ zum Dateianfang (begin)
▪ <code>ios::cur</code>	File-Flag für eine Position relativ zur aktuellen Position (current)
▪ <code>ios::end</code>	File-Flag für eine Position relativ zum Dateiende (end)

- Beispiel: Positionierung des File-Cursors in einem File

```
ios::pos_type pos = file.tellg(); // aktuelle Position merken
file.seekp(pos);                // absolute Positionierung
file.seekp(10, ios::beg);        // rel. Position nach File-Anfang
file.seekp(20, ios::cur);        // rel. Position von aktueller Position
file.seekp(-10, ios::end);       // rel. Position 10 Zeichen vor File-Ende
```

CPP-09

27

Demo: CPP-06-D.05 - FileIO



File I/O Streams: Standardkanäle umleiten



- Durch die Zuweisung der Buffer von Stream-Objekten können die Standard Ein- und Ausgabe-Kanäle in Files umgelenkt werden:

```
void main()
{
    cout << "Zeile vor Umlenkung" << endl;        // screen

    // merken des stream buffers und anlegen eines files
    streambuf* strm_buffer = cout.rdbuf();
    ofstream redirect_file("redirect.txt");

    // Umlenkung durchführen
    cout.rdbuf(redirect_file.rdbuf());
    cout << "Zeile während Umlenkung" << endl;    // file

    // Umlenkung zurückstellen
    cout.rdbuf(strm_buffer);
    cout << "Zeile nach Umlenkung" << endl;        // screen
}
```

CPP-09CPP-06 / 28



06.05

C++ String I/O Streams



- String-Streams werden verwendet um Stream Semantik und Syntax auch für String-Typen anwenden zu können (Include `<sstream>`).

- Beispiel: Double - String Konvertierung und umgekehrt

```
string Double2String(double num) {
    ostringstream sstr;
    sstr << setprecision(2) << fixed << showpoint << num;
    return sstr.str();
}

double String2Double(const string& str) {
    double num;
    istringstream(str) >> num;
    return num;
}
```

CPP-09

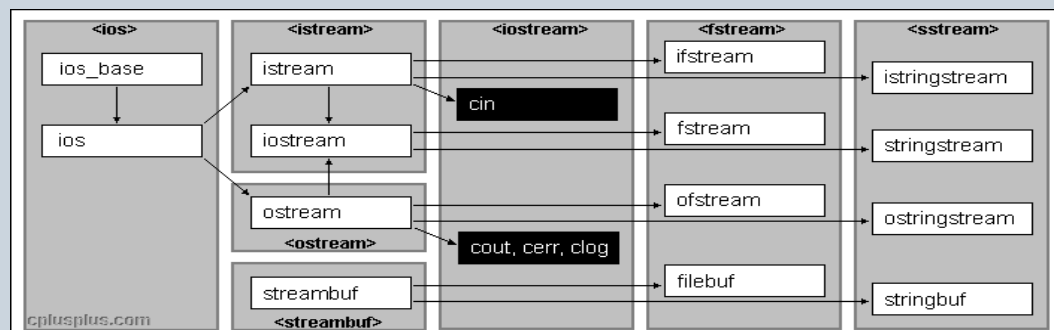
29

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

I/O Streams - Zusammenfassung



- Überblick der vordefinierten Stream Objekte und Typen (typedef)
(Diese sind jeweils in einer `char` und einer `wchar_t` Variante definiert.)



CPP-09

30

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Standard Library Utilities



- Im Header File `<utility>` der Standard Library werden neben allgemein verwendbaren Funktionen der Datentyp `pair` deklariert:
- Datentyp `pair`
 - Die Standardbibliothek verwendet an einigen Stellen Wertepaare (z.B. `Map`, `Multimap`), für welche ein entsprechender Template Datentyp `pair` deklariert wird:

```
template<class T, class U> struct pair {  
    T first;                // first data member  
    U second;              // second data member  
    pair();                // default constructor  
    pair(const T& x, const U& y); // init constructor  
};
```

CPP-09 31

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Standard Library Utilities II



- Im Header File `<algorithm>` der Standard Library werden die folgenden allgemein verwendbaren Hilfsfunktionen definiert:
- Hilfsfunktionen
 - Die `min()` und `max()` Template-Funktionen können verwendet werden um das Minimum bzw. Maximum zweier Werte eines beliebigen Datentyps zu ermitteln (Voraussetzung: Definition eines entsprechenden Vergleichsoperators `<`).
 - Die `swap()` Template-Funktion kann zum Vertauschen zweier Werte eines beliebigen Datentyps verwendet werden.

CPP-09 32

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Standard Library Utilities III



- In den Header Files `<cstdint>` und `<cstdlib>` werden, wie in den entsprechenden C Header Files, häufig verwendete Konstanten, Typen und Makros definiert:
- `<cstdint>`
 - `NULL` Zeigerwert "nicht definiert" (In C++ ist dieser Wert: `int 0`)
 - `size_t` Datentyp für Grössenangaben
 - `ptrdiff_t` Datentyp für Zeiger-Abstände
 - `offsetof()` Offset einer Komponente in einer Struktur oder einer Union
- `<cstdlib>`
 - `exit(int status)` Programm beenden
 - `abort()` Programm abbrechen
 - `atexit(void (*func)())` Funktion welche durch `exit()` aufgerufen werden soll

CPP-09

33

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Standard Library Utilities IV



- Im Header File `<limits>` werden die (plattformabhängigen) numerischen Limiten der verschiedenen Datentypen durch die Template Klasse `numeric_limits` zur Verfügung gestellt:
- Deklaration von `numeric_limits`

```
template<class T> class numeric_limits {
public:
    static T max() throw();
    static T min() throw();
    static const bool is_integer;
    static const bool is_modulo;
    static const bool is_signed;
    ...
};
```
- Beispiel:

```
cout << "max(long): " << numeric_limits<long>::max() << endl;
```

CPP-09

34

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Weitere Standard Library Klassen



- String-Stream-Klassen für die Ein- und Ausgabe von C++ Strings
- Bibliothek für die Verarbeitung von komplexen Zahlen `<complex>` und für numerische Arrays `<valarray>` zur Vektor und Matrizen-Berechnung
- Template-Klasse für die Verarbeitung von Bitsets
- Klassen für die Internationalisierung