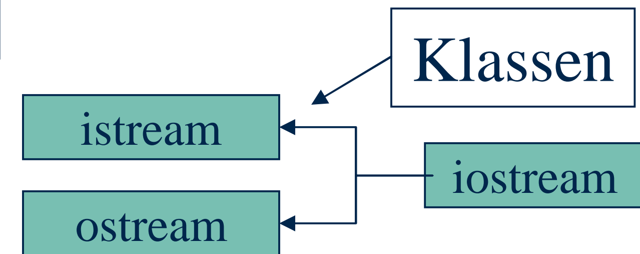
„Streams“

# Überblick

- Rückblick Standard I/O Klassen
- Dateien als F-streams
  - Öffnen von Dateien
  - Lesen und Schreiben
  - Sonderbefehle
  - Unterschiedliche Datei-Typen
- Der C-Stil – für (un)gepufferter Zugriff
- Ein kurzer Vergleich

# Standard I/O - Die Stream Ein/Ausgabe (Klassen)

Include




- Standard <iostream> liefert
  - cin Standardeingabe
  - cout Standardausgabe
  - cerr Standardfehlersausg.
  - clog gepufferte Fehlerausg.

  Stehen automatisch (durch das inkludieren zur Verfügung.
- Gepuffert / ungepuffert
  - Nur cerr schreibt sofort sonst erfolgt Pufferung.
- Normalerweise sind alle Ausgaben mit dem Bildschirm verbunden.
  - Lassen sich aber auch umlenken
  - z.B. bei Programmstart
    Mein_Pro >test.out
    Mein_Pro 2>error.out

```
#include <iostream>
…

int i,n,p=2;

cin >> i;     // Erwartet Integer Eingabe

cout << "i = " << i;    // Ausgabe in
                        // den Puffer

cout << endl;           // Ende der Zeile
                        // & Ausgabe

cout << "Achtung : ";   // erst gepuffert
cerr << "Error ";       // dann aber
                        // sofort Ausgabe
```

# Standard I/O - Manipulatoren & Methoden

- Durch die "<<" ">>" Operatoren werden alle Variablen "formatiert" ausgegeben.
- Zusätzliche Beeinflussung durch:
  - cout << setw(n)…   Manipulator
  - cout.precision(2)   Methoden

  Durch "Methoden" / Manipulatoren
- Durch setzten von Flags (Bits)
  - Cout.setf ()   oder  setiosflags()

```
#include <iostream>
…

double f = 1234.12345678901234;

cout.setf (ios::scientific);
cout.unsetf (ios::fixed);
cout.precision (4);
cout << f << endl;   // 1.2341e+03

cout.setf (ios::fixed, ios::floatfield);
cout.precision (8);
cout << f << endl;   // 1234.12345679
```

2 Parameter = Maske
- löscht automatisch die abh. Flags
  basefield :  oct | dec | hex
  adjustfield : left | right | internal
  floatfield  :  fixed | scientific

ios:scientific

ios:fixed

Maske ios:floatfields

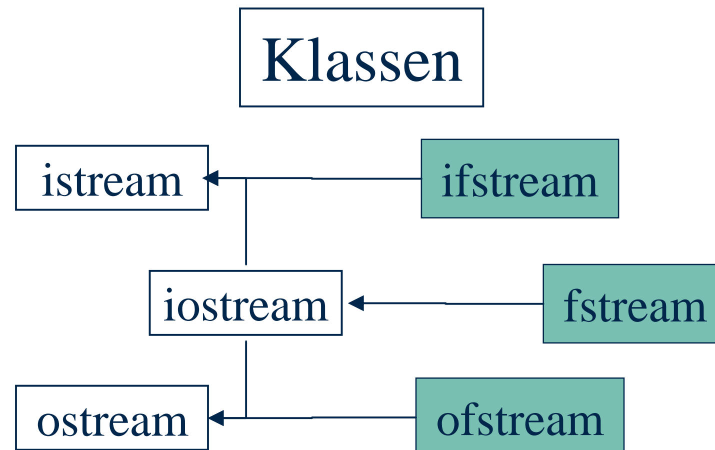
# Standard I/O - Zusatzfunktionen

- Lesen von Zeichen(ketten)
  - cin.getline() lesen einer Zeile
  - cin.get() lesen einer Zeichen(kette)
- Fehlerbehandlung
  - cin.eof() Dateiende ( ^Z )
  - cin.fail() Fehler
  - cin.clear() "Fehler" Status zurücksetzen
  - cin.good(), cin.bad()
- Puffermanipulation
  - cin.putback() zurückschreiben
  - cin.peek() lesen & im Puffer lassen
  - cin.ignore() Zeichen ignorieren

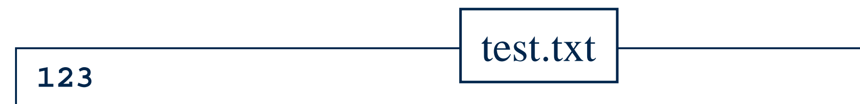
```
#include <iostream>
…

char Zeile[256];
int i;

cin.getline(Zeile,255);

cin >> i;

if (!cin)
  cinClear();  // Rest der Eingabe löschen

…..

void cinClear ()
{
    char Muell;
    cin.clear();// löschen des Status Bits
    do{
     cin.get(Muell);    // nächst. Z. lesen
    }while (Muell!='\n'); // bis Zeilenende
}

oder
cin.ignore(256,'\n');
```

# Datei I/O - Öffnen einer Datei

- Die "Datei"-Klassen sind im <fstream> include
  - Sie erben "alles" von iostream
- Es gibt neue Typen/Klassen
  - ifstream für ein Eingabefile
  - ofstream für Ausgabefile
- Schritte zur Datei:
  - Im Standard ist zunächst nichts geöffnet. D.h. Kein "stream" definiert.
  1. ifstream MeinFile; à Variable/Instanz
  2. MeinFile.open ("Dateiname.txt") à öffnen
  3. MeinFile >> i; à Standard Lesen
- Oder alles in Einem
  - Ifstream MeineDatei("Dateiname.txt");




```
#include <fstream>
…

ifstream MeineDatei ("test.txt");
int i;
MeineDatei >> i
cout << i << endl;  // Ausgabe "123"
```



# Datei I/O - Dateitypen und Optionen zum Öffnen.

- Es gibt unterschiedliche Dateitypen
  - ASCII d.h. Textdateien (z.B. .cpp Datei)
    - Enthalten "text" und Zeilenvorschub,…
  - Binär z.B. die Programmdateien
    - Enthalten "alle möglichen" Bitmuster z.B. 0x00
    - Daten werden in "internem" Format geschrieben
- Typ-Zuweisung beim Öffnen durch Flag
  - MeinFile.open ("Name" , ios:: binary );
    - Standard ist Text Modus
- Zusätzliche Modi sind
  - ios::in, ios::out für Ein-/Ausgabe
  - ios::app An vorhandene Datei anfügen (output)
  - ios::ate Ans Ende der Datei gehen
  - ios::trunc Alles löschen falls mit out geöffnet

Kombination durch "or" MeineDate.open ("Name" , ios::out|ios::trunc);

| bin | in | out | trunc | app/ate |
|---|---|---|---|---|
| | x | | | |
| | x | x | | |
| | x | x | x | |
| | | x | | |
| | | x | x | |
| | | x | | x |
| x | x | | | |
| x | x | x | | |
| x | x | x | x | |
| x | | x | | |
| x | | x | x | |
| x | | x | | x |

# Datei I/O  Schreiben & Lesen von Dateien

- Standard I/O Operatoren
  - MeineDatei << Variable
- Typisches Lesen für Dateien
  - getline() zum Lesen einer Zeile
  - get()  zum Lesen der nächsten Zeichen
- Zusätzliche Schreibfunktionen
  - put ('c')       ein Zeichen
- Binäres Lesen/Schreiben
  - write(char *, n)  Zeiger auf "byte"
  - read (char *, n)  Zeiger auf byte-memory

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

int nchar;           // Zeichen
ofstream outfile;    // Ausgabedatei

outfile.open("test.out", ios::out);
if (outfile.fail())  // Fehler abfangen
{
    cerr << "Fehler beim Öffnen ";
    return 8;
}

for (nchar = 0; nchar < 128; ++nchar)
{
  outfile << nchar;
}

return 0; // Datei wird automatisch geschl.

….
```

123456789101112131415161718192021222324252627282930…..
0121122123124125126127

# Datei I/O- Fehlerbehandlung

- Bei jeder Aktion wird ein Status gesetzt
  - Auslesen mit iostate rdstate()
- Dieser kann auch durch spez. Methoden gelesen
  - eof() Dateiende ( ^Z )
  - fail() Fehler
  - clear() "Fehler" Status zurücksetzen
  - cin.good(), cin.bad()
- Oder manipuliert werden
  - clear() auf OK setzen
  - clear(iostate s) auf Bit "s" setzen
- Einfacher geht's auch durch direktes Abfragen des Streamobjektes
  - if (!MeineDatei)

    Der Status wird dann vom Compiler in bool umgewandelt

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

int nchar;            // Zeichen
ifstream infile;      // Eingabedatei

infile.open("test.out", ios::in);
if (infile.fail())  // Fehler abfangen
{
    cerr << "Fehler beim Öffnen ";
    return 8;
}

char Kette[256];
infile.getline(Kette,255);
if (!infile)
{
          //Fehlerbehandlung  eof ? , …
}
```

# Datei I/O - Navigation in Dateien

- § Zu einer bestimmten Position gehen
  - Ø seekg (p)   zu absoluter Position "p"gehen
  - Ø seekg (p, bezug)   zu relativer Position gehen
    - o ios::beg   von Begin an gezählt
    - o ios::cur oder ios::end
  - Ø Bzw. seekp() für Ausgabe
- § Position ist dabei:
  - Ø Zahl in byte (genauer vom Typ Ios::pos_type)
  - Ø Die Aktuelle (Lese-)Position ist: ios::pos_type Pos = tellg()
  - Ø Schreibposition = tellp()

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

ifstream IS("test.dat");

IS.seekg (5,ios::beg);  // 5 Zeichen vom A.

char c;                 // Zeichen lesen
IS.get( c );            // und eine Pos. weiter

IS.seekg(5,ios::cur);          // 5 weiter
Ios::pos_type P = IS.tellg(); // merken

IS.seekg(-4,ios::cur);         // 4 zurück

..

IS.seekg(P,ios::beg);    // auf P gehen
```

# Der C-Stil – gepufferte Ein/Ausgabe

- Alte c Standard Bibliothek
  - #include <cstdio>
- FILE handle und fprintf statt C++ streams und "<<", ">>" Operatoren
  - FILE *FilePointer
  - FilePointer = fopen("Name", mode)
  - mode = "r" "w" "b" (read/write/binary)
  - Z.B "rb" == binary read
- Ein/Ausgabe mit
  - char c = fgetc(FilePointer) (fputc()) für ein Zeichen
  - fgets(Zeile,n,FilePointer) (fputs) für eine Zeichenkette
  - fprintf für formatierte Ausgabe
  - fscanf für formatierte Eingabe
- Binäre Ein/Ausgabe
  - fread() und fwrite()

```
#include <cstdio>

int i=0;
FILE *FileP;

FileP = fopen ("name.txt", "r");  //lesen

if ( FileP == NULL) // Fehler == NULL
{
   cerr << "error";
   exit (8);
}

while (true)
{
   char c = fgetc ( FileP);   // Zeichen
   if (c == EOF)              // lesen
     break;         // Ende bei EOF
   cout << c;
   i++;
}

cout << "Datei ist " << i << "Zeichen gross";
fclose(FileP);      // Datei schliessen
```

# Der C-Stil – ungepufferte Ein/Ausgabe

- Includes:
  - <sys/types.h>
  - <sys/stat.h>
  - <fcntl.h>
  - <io.h>
- Statt Stream-Objekt
  - Integer "File-Descriptor"
- Öffnen
  - int Fdes = open("Name", O_RDONLY)
    - Flags: Lesen/Schreiben, Bin,….
- Schreiben/Lesen
  - read ( Fdes , Puffer, maxZeichen)
  - write (Fdes, Puffer, Zeichen)

```
int infile;

infile = open ( "name" , O_RDONLY);

if (infile < 0)     // Fehler < 0
{
   cerr << "Fehler ";
   return 8;
}

char buffer[256];   // Lese Puffer
int  insize;        // Gelesene Zeichen

while (true)
{
  insize = read(infile,buffer,  // Rückgabe
           sizeof(buffer));     // # Bytes

  if (insize == 0)  // EOF
    { ….eof …}
  if (insize < 0)   // Fehler
    { -…Fehler …}
  ….
}
close (infile);     // Datei schliessen
```

# Datei I/O - Vergleich der Methoden

- Einfache Verwendung
  - C-Stil ist kompakter
    - printf ("%2d/%2d/%2d,tag,monat,jahr); à 01/12/03
  - C++ Stil
    - cout << setw(2) << tag << '/' << setw(2) << …….
  - Aber manchmal ist printf etwas verwirrend.
  - Bei Verwendung der Klassen (überladen) lässt sich vieles vereinfachen
    - cout << datum;
- Zuverlässigkeit
  - Achtung gets (buffer) liest alles – egal ob genügend Platz für buffer beschafft wurde (buffer[128]).
  - Häufiger Fehler: falsche Argumente bei printf()…