Herzlich willkommen!

Dozent: Dipl.-Ing. Jürgen Wemheuer

Mail: wemheuer@ewla.de
Online: http://cpp.ewla.de/

Das EVA-Prinzip

hier rein ->

-> da raus

E (Input) -> V (Processing) -> A (Output)

Programmiertechnische Möglichkeiten für die Eingabe:

immer möglich:
Übergabe von Parametern beim Aufruf des Programms, z.B.:

```
prog.exe Var1 Var2 Var3
```

Dann Entgegennahme in der main-Funktion:

```
int main(int argc, const char *argv[])
// Anzahl (4), Array der Parameter: argv[0] =("prog.exe“) …
```

Im klassischen C:

```
scanf("%[typ]",&x); //typ ersetzen durch gewünschten Datentyp
```

In C++:

```
cin >> x; // nennt sich „Input-Stream“
```

„Drucken, Scannen und Streamen“

Programmiertechnische Möglichkeiten für die Ausgabe:

immer möglich:
Rückgabe eines Exit-Werts am Ende der main-Funktion

```
return 404;
```

(wird nicht angezeigt, muss „jemand“ auswerten…)

Im klassischen C:

```
printf("Ausgabe der Werte a:%d, b:%3.2f c:%c", a, b, c);
```

In C++:

```
cout << x; // nennt sich „Output-Stream“
```

printf(…)-Bedienungsanleitung

- benötigt die Bibliothek für das Standard-Ein-Ausgabe-Gerät
- braucht als erstes Argument IMMER eine Zeichenkette
- kann mehrere Parameter verarbeiten, die durch Kommata getrennt der Reihenfolge nach eingesetzt werden
- Beispiel:

```
#include <stdio.h>
int main() {
  printf ("%i plus %i ist gleich %s.\n", 3, 5, „Acht");
  return 0;
}
```

- Nachteil: Umständliche Handhabung (Parameter, Datentypen,…)
- Vorteil: Bequeme Formatierung von Gleitkommazahlen…

Format-Platzhalter für printf()

%i	Ganzzahl (integer)
%d	Dezimalzahl = Ganzzahl (genau wie %i)
%ld	„long int“-Ganzzahl
%u	„unsigned int“ – als positive Zahl ohne Vorzeichen
%f	Fließkommazahl
%e	Fließkommazahl in wissenschaftlicher Darstellung
%g	Fließkommazahl (%e oder %f, was besser passt…)
%x	Ganzzahl in Hexadezimaldarstellung
%o	Ganzzahl in Oktaldarstellung
%c	Zeichen (character)
%s	Zeichenkette (string)
%%	das Prozentzeichen selbst muss maskiert werden
%p	Adresse eines Wertes (Pointer, Zeiger)
%3.2f	zusätzliche Parameter für die Formatierung möglich

cout - Bedienungsanleitung

- benötigt die Bibliothek für die Standard-Datenströme
- kann mehrere Parameter verarbeiten, die durch den Ausgabeoperator **<<** in den Ausgabestrom eingefügt werden
- Beispiel:

```
#include <iostream>
using namespace std;
int main() {
  int I=3; double D=5.0; string S= "Acht";
  cout << I << " plus " << D << " ist gleich " << S << ". " << endl;
return 0;
}
```

- Vorteil: Intuitive Handhabung
- Nachteil: Umständliche Formatierung der Anzeige - aber gegenüber dem printf() mehr Möglichkeiten („Manipulationen“)

scanf(…) - Bedienungsanleitung

- benötigt die Bibliothek für das Standard-Ein-Ausgabe-Gerät
- kann immer nur EINE (1) durch <Enter> abgeschlossene Eingabe verarbeiten
- Beispiel:

```
#include <stdio.h>
int main(void) {
    int Ganzzahl;
    scanf("%i", &Ganzzahl);        // in einen Speicher schreiben
    printf("Inhalt der Speicherzelle: %d\n", Ganzzahl);
    return 0;
}
```

- Nachteil: Umständliche, fehleranfällige Handhabung
- Vorteile: keine…

Format-Platzhalter für scanf()

%d	vorzeichenbehafteter Integer als Dezimalwert
%i	vorzeichenbehafteter Integer als Dezimal-, Hexadezimal oder Oktalwert
%e	Fließkommazahl
%f	Fließkommazahl
%g	Fließkommazahl
%o	Integer als Oktalzahl einlesen
%s	Zeichenkette einlesen
%x	Zeichen als Hexadezimalwert einlesen
%%	erkennt das Prozentzeichen

cin - Bedienungsanleitung

- benötigt die Bibliothek für die Standard-Datenströme
- kann immer nur EINE (1) durch <Enter> abgeschlossene Eingabe verarbeiten
- Beispiel:

```
#include <iostream>
using namespace std;
int main() {
    int Ganzzahl;
    cin >> Ganzzahl;
    cout << "Meine Ganzzahl lautet: " << Ganzzahl << endl;;
    return 0;
}
```

- Vorteil: Intuitive Handhabung
- Nachteile: keine…

Escape-Sequenzen

\a (alert) = gibt einen hör- oder sichtbaren Alarm aus, ohne die Position des Cursors zu ändern.

\b (backspace) = Setzt den Cursor ein Zeichen zurück. Wenn sich der Cursor bereits am Zeilenanfang befindet, dann ist das Verhalten unspezifiziert.

\f (form feed, „Blattvorschub“) = Setzt den Cursor auf die Startposition der nächsten Seite.

\n (new line, „Zeilenvorschub“) = Setzt den Cursor auf die Anfangsposition der nächsten Zeile.

\r (carriage return, „Wagenrücklauf“) = Setzt den Cursor an den Zeilenanfang.

----- *Beachte die unterschiedliche Interpretation bei Windows-, Linux und Apple-Rechnern* -----

\t (horizontal tab) = Setzt den Tabulator auf die nächste horizontale Tabulatorposition. Wenn der Cursor bereits die letzte Tabulatorposition erreicht hat, dann ist das Verhalten unspezifiziert (vorausgesetzt eine letzte Tabulatorposition existiert).

\v (vertical tab) = Setzt den Cursor auf die nächste vertikale Tabulatorposition. Hat der Cursor bereits die letzte Tabulatorposition erreicht, dann ist das Verhalten unspezifiziert (wenn eine solche existiert).

\" Das Zeichen " wird ausgegeben

\' Das Zeichen ' wird ausgegeben

\? Das Fragezeichen ? wird ausgegeben

\\ Das Zeichen \ wird ausgegeben

\0 Ende-Markierung einer Zeichenkette

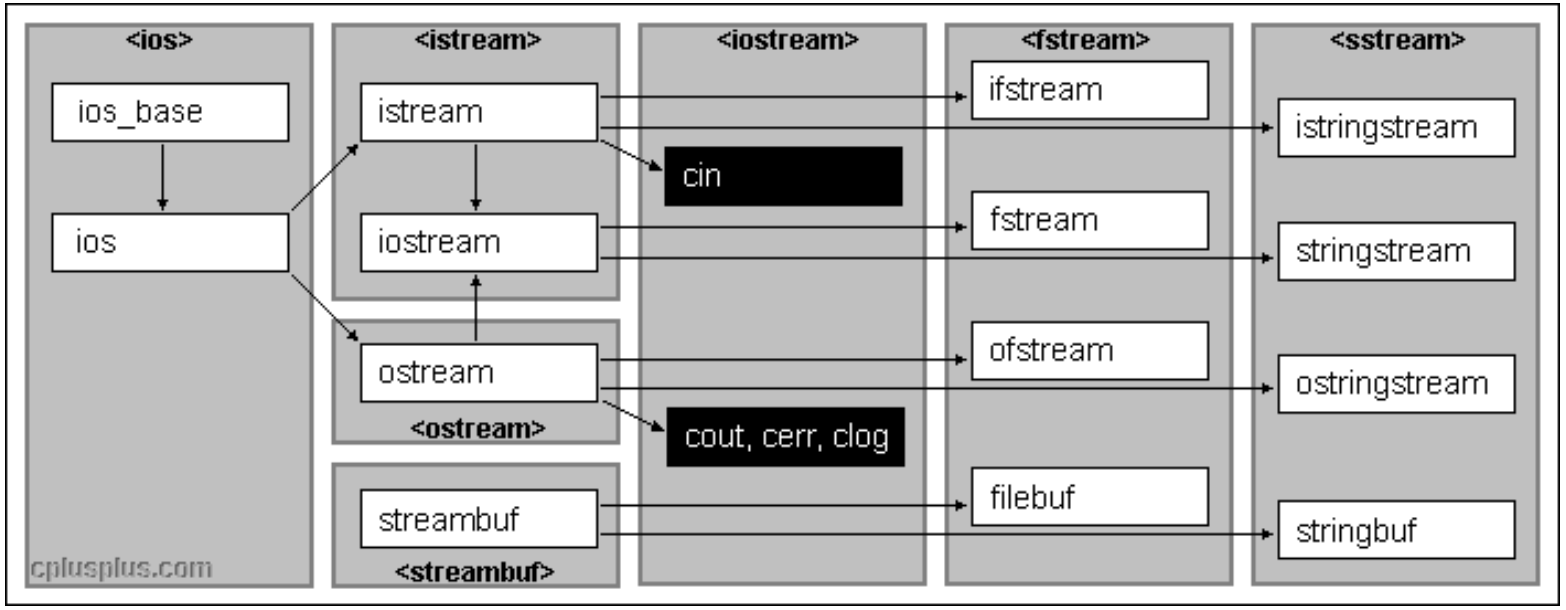
getchar() und putchar(.)

- Die Funktion getchar() liefert das nächste Zeichen von der Standardeingabe.
  - Weil die Standardeingabe üblicherweise zeilenweise und nicht zeichenweise eingelesen wird, muss man nach der Eingabe des Zeichens üblicherweise <Enter> drücken, damit überhaupt etwas passiert.
- Die Funktion putchar(c) gibt ein einzelnes Zeichen c auf der Standardausgabe aus.

```
char c;
c = getchar();
putchar(c);
```

- sind in der Bibliothek <stdio.h> / <cstdio> enthalten

Das Stream-Konzept in C++



Variation der cin-Eingabe

- cin liest den Strom des Eingabegeräts bis zum nächsten, als Trennzeichen interpretierbaren Zeichens (z. B. ein Leerzeichen), der Rest bleibt im Eingabepuffer (Tastaturpuffer).
- Will man eine ganze Zeile in eine Zeichenkette einlesen, verwende man besser:

```
string Text;
cin.sync();          // evtl. Tastaturpuffer leeren
getline(cin, Text);
```

- optional kann man ein Trennzeichen angeben, dann wird die Eingabe nur bis zum ersten Auftreten dieses Zeichens ausgewertet:

```
getline(cin, Text, 'x');
```

- Alternativ / zusätzlich lässt sich auch die Länge der Eingabe begrenzen:

```
cin.getline(Text, 20);
cin.getline(Text, 20, 'x');
```

Manipulation der cout-Ausgabe

- Recht umfangreich lässt sich in C++ der cout-Ausgabestrom „manipulieren“, benötigt wird dazu die Bibliothek <iomanip>
- die vollständige Handhabung ist jedoch „ohne Spickzettel“ kaum zu beherrschen…
- Formatangaben werden:
  - einmal gesetzt und bleiben so erhalten, bis eine Änderungsanweisung erscheint
  - einmal gesetzt und gelten NUR für die unmittelbar folgende Ausgabe
  - separat in einer eigenen Anweisung gesetzt oder
  - direkt in den cout << Ausgabestrom eingefügt
  - manchmal NUR in Kombinationen sinnvoll (z.B.: setw / right)
- Immerhin: die wichtigsten Manipulationen sind recht einfach umzusetzen.

Input-Output-Manipulationen

Die Bibliothek <iomanip> bietet folgende Manipulationen, die als Schalter dauerhaft für jeden Strom gesetzt werden:

Set Flag	Unset Flag	Beispiel
`boolalpha`	`noboolalpha`	`cout << boolalpha << true;`
`showbase`	`noshowbase`	`cout << hex << showbase << zahl;`
`showpoint`	`noshowpoint`	`cout << showpoint << 5; // 5.`
`showpos`	`noshowpos`	`cout << showpos << 5; // +5`
`skipws`	`noskipws`	`cout << skipws << "  123"; // 123`
`unitbuf`	`nounitbuf`	`cout << unitbuf << …; // Puffer leer`
`uppercase`	`nouppercase`	`cout << uppercase << "0X4D"; // 0x4d`
`dec/hex/oct`	`---`	`cout << hex << 70;  // 46`
`fixed/scientific`	`---`	`cout << scientific << 2015; // 2.01500+003`
`internal/left/right`	`---`	`cout << right << 5;  // 5`

Input-Output-Manipulatoren

Die Bibliothek <iomanip> bietet folgende Manipulatoren, die mit Parametern in den Strom eingefügt werden:

Manipulator	Bedeutung	Beispiel
setiosflags	Setzt dauerhafte Einstellungen	cout << setiosflags(…) << …;
resetiosflags	Löscht dauerhafte Einstellungen	cout << resetiosflags(…) << …;
setbase	Dezimal-, Hex-, Oktal-Anzeige	cout << setbase(16) << zahl;
setfill	Füllzeichen bestimmen	cout << setfill('0') << zahl;
setprecision	Anzahl der Ziffernstellen	cout << setprecision(5) << …;
setw	Breite setzen	cout << setw(5) << zahl;
get_money	Eingabe als Währung auswerten	cin >> get_money(preis);
put_money	Zahl als Währung ausgeben	cout << put_money(10.50L);
get_time	Eingabe als Zeit auswerten	cin >> get_time(&zeit,"%R");
put_time	Zahl als Zeit ausgeben	cout << put_time(zeit,"%c");

Datei-Operationen mit fprintf(…)


```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   FILE * fp;
   fp = fopen ("file.txt", "w+");            // Datei zum Schreiben öffnen
   fprintf(fp, "%s %s %s %d", "Wir", „leben", "in", 2016);
   fclose(fp);

   int c;
   fp = fopen("file.txt","r");               // Datei zum Lesen öffnen
   while(true) {
      c = fgetc(fp);
      if (feof(fp)) { break; }
      printf("%c", c);
   }
   fclose(fp);
   return 0;
}
```

Datei-Operationen mit <fstream>


```
#include <iostream>
#include <fstream>                      // <ifstream> / <ofstream>
using namespace std;

int main() {
    string Eingabe;
    ifstream fin;                       // Lesen aus einer Datei
    fin.open(“datei.txt“, ios::in);
    fin >> Eingabe;

    ofstream fout;                      // oder: fstream fout ("test.dat", ios::out);
    fout.open("test.dat", ios::out);    // absolute oder relative Pfad-Adresse
    fout << "Dieser Text geht in die Datei" << endl;
    fout.close();
    return 0;
}
```

system("…")

Der system()-Befehl führt eine Anweisung auf der Kommandozeilen-Ebene des jeweiligen Betriebssystems aus! Also mit Vorsicht anwenden!

Beispiele:

```
system ("format C:");          // „Platte putzen“
system ("del *.*");            // alle Dateien löschen
system („cls");                // Bildschirm leeren
system („color 2A");           // Farben setzen
system ("C:\\Programs\MSOffice\excel.exe");
```