P. S. offene Fragen

1. mehrfache Deklarationen in Klassen?

```
class X {
    void foo();
    void foo() { .... }
};
```

Nein!

Memberfunktionen können in der Klasse (genau einmal) deklariert oder definiert werden. Nur wenn nur deklariert wurde, darf außerhalb der Klasse (nur) definiert werden.

2. Warum keine Kontollflussanalyse in C++?

- `exit`, Exceptions http://www.gotw.ca/gotw/020.htm, `asm`-Einschlüsse
- dennoch: § 6.6.3: Flowing off the end of a function is equivalent to a return with no value; this results in undefined behavior in a value-returning function. -- in C legal wenn Wert nicht benutzt.

  http://stackoverflow.com/questions/1610030/why-can-you-return-from-a-non-void-function-without-returning-a-value-without-pr

1.4. Funktionen

- können `inline` sein: kein Aufruf, sondern (Seiteneffekt-freie und typgerechte) Substitution auf Quelltextebene:

```
inline int square(int i){return i*i;}
int main()  { std::cout << square(4); }
```

inline substitution

```
int main()  { std::cout << 4*4; } // u.U. sogar 16
```

- Ziel: Effizienz, auch wenn call overhead > 'Nutzeffekt' der Funktion
- Memberfunktionen, die im Klassenkörper definiert werden, sind implizit `inline` ! (gute Kandidaten, weil meist kurz)

Tony Hoare: "Premature optimization is the root of all evil ! "

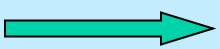
siehe auch
www.ddj.com (search for: inline redux) und www.gotw.ca/gotw/033.htm

1.4. Funktionen

– können default arguments haben: ein Endstück der Argumentliste einer Deklaration mit Wertevorgaben

```
int atoi (const char* string, int base = 10);
// ascii to int on radix base
atoi ("110"); // --> atoi("110", 10) --> 110
atoi ("110", 2); // --> atoi("110", 2) --> 6
```

Vorsicht Falle 1: `void foo(char*=0);`



`void foo(char* =0);`

1.4. Funktionen

Vorsicht Falle 2:

```
int f(int);
int f(int, int=0);
f (1); // mehrdeutig: f(1) oder f(1,0)
```

- variable Argumentlisten a la `printf` in C++: `...` ellipsis

```
extern "C" int printf (const char* fmt, ...);
```

`extern "C"` ist eine sog. linkage Direktive: hier kein name mangling

1.4. Funktionen

– können überladen werden: gleicher Name, unterscheidbare Signatur (Rückgabetyp spielt KEINE Rolle!)

name mangling

```
class X{ public:
   X();                          __1X
   X(int);                       __1Xi
   int foo();                    foo__1X
   int foo() const;              foo__C1X
   int foo(const X&);            foo__1XRC1X
};
int foo(int);                    foo__Fi
double foo(double);              foo__Fd
void foo(char*, int);            foo__Fpci
int printf(const char*, ...);    printf__FPCce
```

```
$ g++ -c foo.cc
$ nm foo.o
00000000 W __1X
00000000 W __1Xi
....
$ nm foo.o | c++filt
00000000 W X::X(void)
00000000 W X::X(int)
....
```

1.5. Strukturierte Anweisungen

(fast) wie in Java:

while, do, for, if, switch, break, continue, return

Deklaration in Blöcken sind Anweisungen: Deklaration von Objekten am Ort des Geschehens (wie in Java)

```
void foo()
{
      int i=0;
      bar(i); ....
      int j=3;
      bar(j); ....
}
```

Vorsicht Falle:

```
if (x=1) ....
```

neu in C++11: range-based for

```
int array[] = { 1, 2, 3, 4, 5 };

for (int x : array) // value
    x *= 2;

for (int& x : array) // reference
    x *= 2;
```

C++11

Ersetzung durch:

```
{
    auto && __range = range-init;
    for ( auto __begin = begin-expr,  __end = end-expr;  __begin != __end;  ++__begin ) {
        for-range-declaration = *__begin;
        statement
    }
}
```