Einfaches Demonstrationsbeispiel zu Speicherklassen

- *main.cpp:*

```
#include <iostream>
#include <cerrno>
#include <cstring>

using namespace std;

extern double ld(double); /* alternativ: Dekl. in ld.h unterbringen und
                                         mit #include "ld.h" einlesen */

int main()
{
  double x;
  cout << "x: ";
  cin  >> x;
  errno = 0;
  cout << "ld(" << x << ") = " << ld(x) << endl;
  if (errno) cout << strerror(errno) << endl;
  return 0;
}
```

ld.cpp:

```
extern "C" double log(double x);

static double ln2 = 0.69314718055994530942;

extern double ld(double x)
{
  return log(x)/ln2;
}
```

- *Übersetzen:* `c++ main.cpp ld.cpp`
- *Ausgabe:*

```
x: 8
ld(8) = 3

x: 0
ld(0) = -inf
Numerical result out of range

x: -4
ld(-4) = nan
Numerical argument out of domain
```

- *Linkersymbole (Ausgabe von* `nm` *unter Ubuntu Linux 18.04 amd64*

```
main.o:
                   :
                 U __errno_location
                   :
0000000000000000 T main
                   :
                 U strerror
                 U _Z2ldd
                   :
                 U _ZSt3cin
                 U _ZSt4cout
                   :
ld.o:
                 U log
0000000000000000 T _Z2ldd
0000000000000000 d _ZL3ln2

a.out:  (Auszug)
                   :
                 U __errno_location@@GLIBC_2.2.5
                   :
                 U log@@GLIBC_2.2.5
0000000000000b0a T main
                   :
                 U strerror@@GLIBC_2.2.5
                   :
0000000000000c7a T _Z2ldd
                   :
0000000000202010 d _ZL3ln2
                   :
0000000000202140 B _ZSt3cin@@GLIBCXX_3.4
0000000000202020 B _ZSt4cout@@GLIBCXX_3.4
                   :
```

Symboltyp	*Bedeutung* *(Auszug)*
`U`	undefiniertes Symbol
`T`	Symbol befindet sich im text-segment (Programmcode)
`D`	Symbol befindet sich im data-Segment (initialisierte Daten)
`B`	Symbol befindet sich im bss-Segment (uninitialisierte Daten)

  - Symboltypen mit Großbuchstaben bezeichnen externe Symbole; dagegen werden interne Symbole mit Kleinbuchstaben geschrieben.
  - Das Vorhandensein undefinierter Symbole in der gebundenen (gelinkten) Objektcodedatei `a.out` hängt mit der Verwendung dynamischer Bibliotheken zusammen (Voreinstellung).

Überladen und Binden

Damit beim Binden gleichnamige Funktionen unterschieden werden können, werden von vielen Übersetzern im Symbolnamen die für die Unterscheidung notwendigen Informationen mitabgespeichert.

Bsp.:

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

float sqr(float x)
{ return x*x; }

double sqr(double x)
{ return x*x; }

complex<double> sqr(complex<double> z)
{ return z*z; }
```

Verkürzte und umsortierte Ausgabe: `CC -c sqr.cpp; nm -og sqr.o` *(CC-6.2 Solaris 10)*

```
....       Type  Bind  Other Shndx   Name

....      |FUNC |GLOB |0    |2      |__1cDsqr6Ff_f_
....      |FUNC |GLOB |0    |2      |__1cDsqr6Fd_d_
....      |FUNC |GLOB |0    |2      |__1cDsqr6FnDstdHcomplex4Cd___1_
```

Stark verkürzte und umsortierte Ausgabe: `g++ -c sqr.cpp; nm -og sqr.o` *(g++-3.4 Solaris 10)*

```
....       Type  Bind  Other Shndx   Name

....      |FUNC |GLOB |0    |2      |_Z3sqrf
....      |FUNC |GLOB |0    |2      |_Z3sqrd
....      |FUNC |GLOB |0    |2      |_Z3sqrSt7complexIdE
```

Weniger als bei C kann bei C++ davon ausgegangen werden, dass sich die Objektcodedateien und -bibliotheken von *verschiedenen* Compiler zu einem ausführbaren Programm binden lassen, auch wenn es sich um den gleichen Rechner und das gleiche Betriebssystem handelt (*hier ausnahmsweise Solaris*).

Binden mit anderen Programmiersprachen

Obwohl C++ weitgehend abwärtskompatibel zu C ist, erfordert das Zusammenbinden von C- und C++-Objektcodedateien zusätzliche Maßnahmen.

Verkürzte Ausgabe (nur double-Funktion): `cc -c sqr.c; nm -og sqr.o` *(cc-6.2 Solaris 10)*

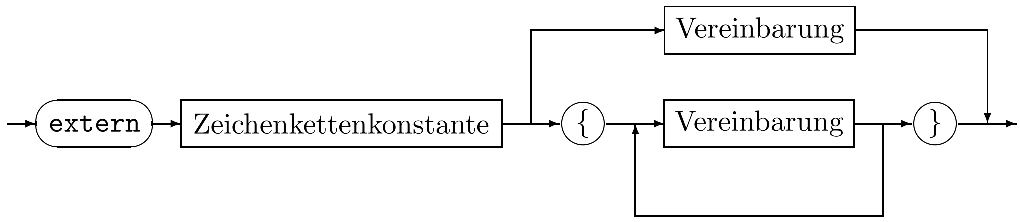
```
....       Type  Bind  Other Shndx   Name

....      |FUNC |GLOB |0    |2      |sqr
```

Verkürzte Ausgabe (nur double-Funktion): `gcc -c sqr.c; nm -og sqr.o` *(g++-3.4 Sol. 10)*

```
....       Type  Bind  Other Shndx   Name

....      |FUNC |GLOB |0    |2      |sqr
```



Die Anweisung darf nicht in Funktionen vorkommen und verändert die Gültigkeitsbereiche nicht. Unterstützt werden zumindest `"C"` und `"C++"`. Letzteres ist voreingestellt.

Bsp.: Berechnung von $\Gamma(x) = \int\limits_0^\infty t^{x-1} e^{-t} dt$ *(Übersetzen:* `c++ gamma.cpp -lgsl -lgslcblas`*)*

```
extern "C" {
#include <gsl/gsl_sf.h>        // GNU Scientific Library special functions
}                              // Ubuntu Linux 18.04, g++-7.5

#include <iostream>
using namespace std;

int main()
{
  double x;
  cout << "x ? ";
  cin >> x;
  cout << "gamma(" << x << ") = " << gsl_sf_gamma(x) << endl;
  return 0;
}
```

Bem.: Die `extern "C"`-Anweisung ist hier nicht notwendig, weil sie in der Headerdatei durch Präprozessormakros für die betreffenden Funktionen automatisch eingefügt wird.