Einfache Programmentwicklung

Werkzeuge

```
gcc      # GNU C-Compiler
g++      # GNU C++-Compiler
ld       # Programmverbinder
make     # Generierung von Programmen
```

Übersetzen eines C-Programms

```
gcc -c hello.c
```

Übersetzen eines C++-Programms

```
g++ -c hello.cc          # oder
g++ -c hello.C           # Solaris oder cxx, cpp
```

Übersetzen und Verbinden zum ausführbaren Programm

```
gcc hello.c              # Ergebnis ist a.out
g++ hello.cc -o hello    # Ergebnis ist hello
```

"Übersetzen" mit `make`

```
make hello               # führt bei einer C Datei zu
> gcc -o hello hello.c
make hello               # führt bei einer C++ Datei zu
> g++ -o hello hello.cc
```

Was ist wenn beide Dateien existieren? (siehe interne Regeln `make`)

- Ausführen der C-Übersetzung

Headerfiles (compilerabhängig/version)

```
<header>                    // Systemheader
"header"                    // Userheader
```

Old-Style

```
#include <stdio.h>          // C-I/O-Header
#include <iostream.h>       // C++- I/O-Header
```

- Namen in den Bibliotheken sind nicht in **`namespace`** eingebunden.
- Wird von allen Compilern (auch vor dem Ansi C++ Standard) verstanden.

New-Style

```
#include <cstdio>          // C- I/O-Header
#include <iostream>        // C++- I/O-Header
```

- Namen der Bibliotheken sind im **`namespace`** `std` eingebunden.
- Minimum ist `2.95.x,` aktuell `4.5`
- Benutzung der Old-Style Header möglich (Zugriff auf Bibliotheken ohne **`namespace`**)

Modularisierung eines Mini-Projektes

Ausgangspunkt:

- Hauptprogramm : `Demo.c`
- Unterprogramm 1 : `Test1.c`, Schnittstelle `Test1.h`
- Unterprogramm 2 : `Test2.c`, Schnittstelle `Test2.h`

Aufgabenstellung:

- Übersetzen des Moduls `Test1` zum Objektmodul `Test1.o`
- Übersetzen des Moduls `Test2` zum Objektmodul `Test2.o` und Einfügen in eine Bibliothek `libTest.a`
- Erzeugen des vollständigen Programmes aus `Demo.c`, `Test2.o` und `libTest.a`

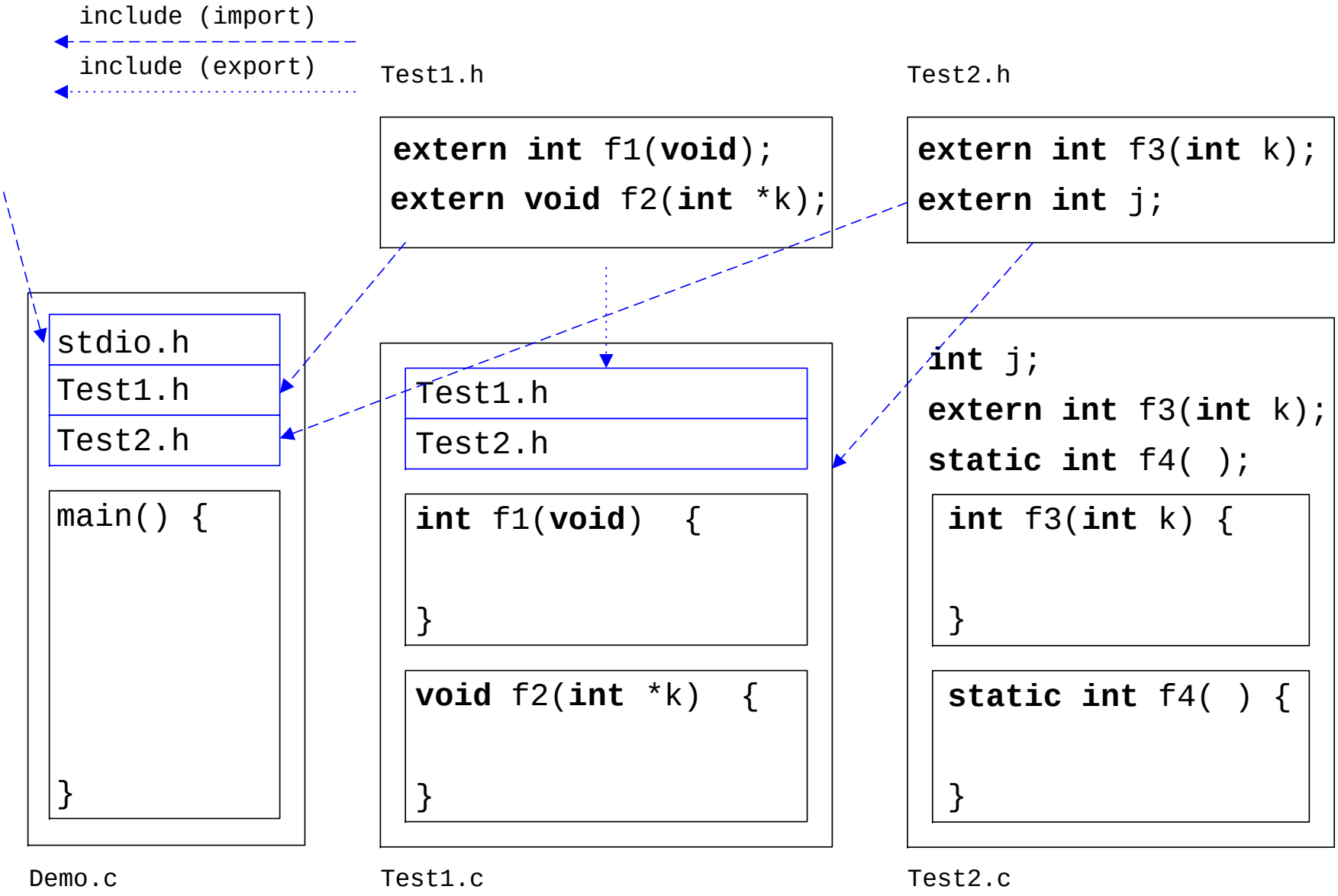
Notwendige Compiler/Linker Flags

`-E`	:	Quelltext nur vorübersetzen
`-S`	:	Quelltext nur übersetzen
`-c`	:	Quelltext nur übersetzen und assemblieren
`-I`	:	Suchpfad für Header Dateien
`-l`	:	Suchpfad für Bibliotheken
`-L`	:	Angabe der Bibliothek
`-O`	:	Optimierung
`-D`	:	Parameter für den Vorübersetzer
`-g`	:	Debug Informationen setzen
`-pg`	:	Profile Informationen setzen
`-o`	:	Ausgabe Datei

Archivar Flags

`-r`	:	Ersetzen oder Einfügen
u	:	Ersetzen, nur wenn einzu-
	:	fügende Datei neuer ist

Beziehungen zwischen den Quelltextdateien



Quelltexte der Beispiel Anwendung

Demo.c

```c
#include <stdio.h>
#include "Test1.h"  /* import     */
#include "Test2.h"  /* import     */

main( )  {
/* komplette Signatur
extern int main(int argc,char** argv);*/

  int i; /* lokale auto Variable */

  printf("Hauptprogramm\n");

  i = f1( );
  /* Aufruf Funktion aus Test1   */
  f3(i);
  /* Aufruf Prozedur aus Test2   */
  printf("i = %d\n", i);
  return 0;
}
```

Test1.h

```c
/*  Header Test1.h                  */
extern int f1(void);
extern void f2(int *l);
```

Test2.h

```c
/*  Header Test2.h                  */
extern int f3(int k);
extern int j;    /* Deklaration  */
```

Test1.c

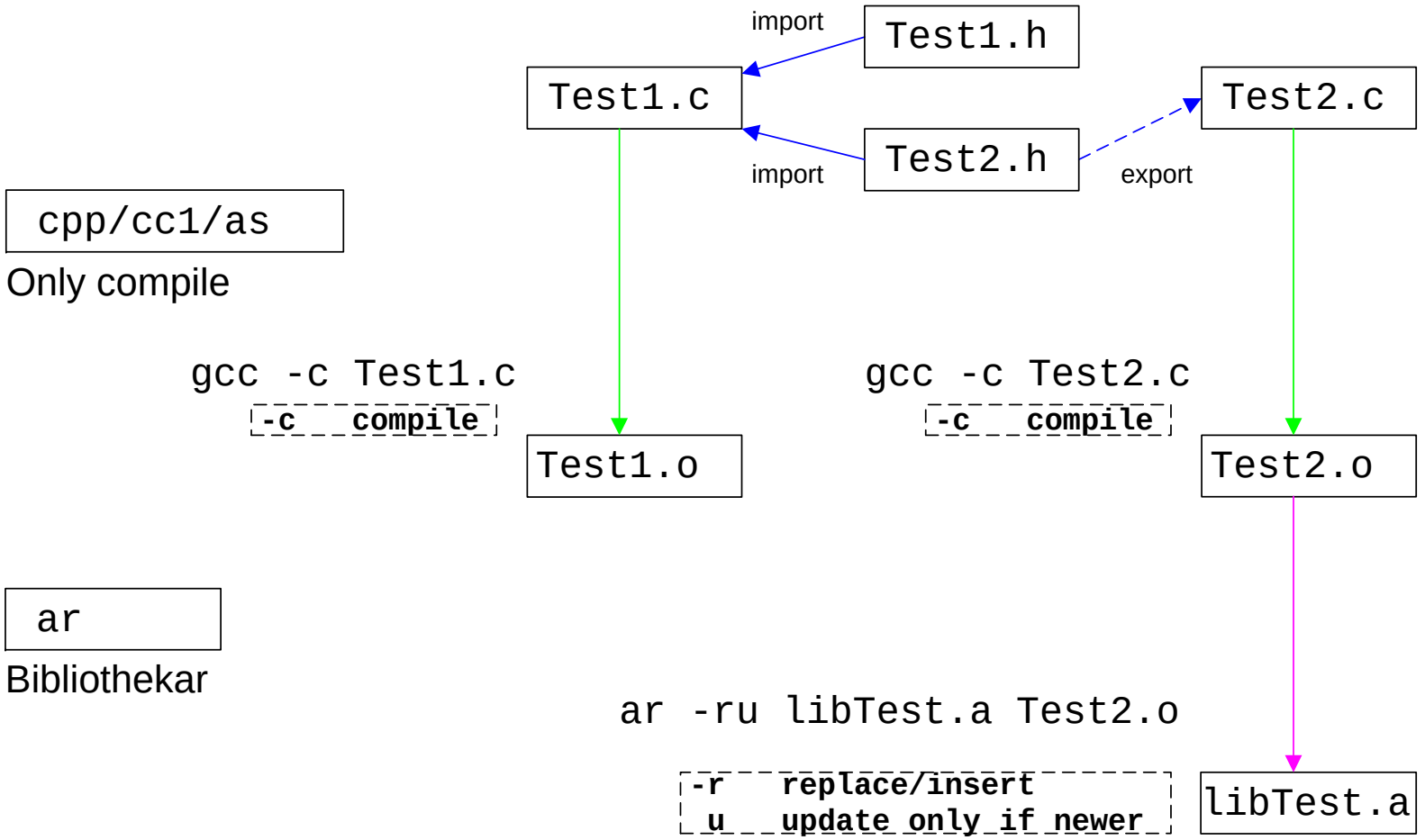
```c
/*  Modul Test1.c                */
#include <stdio.h>
#include "Test1.h"  /* export */
#include "Test2.h"  /* import */

int f1(void)  {

  register int i;

  printf("Aufruf von f1( )\n");
  for( i = 0; i < 10; i ++) {

    j = j + i;
  }
  return i;
}

void f2(int *k)  {

  int j = f1();
  /* verdeckt globales j     */

  printf("Aufruf von f2( )\n");
  *k + f3(j);
}
```

Test2.c

```c
/*  Modul Test2.c                */
#include <stdio.h>

static int f4(void);
static int x = 1;
int j; /* Definition global Var.*/

int f3(int k)  {
/* extern int f3( int)           */

  int i;  /* auto Variable i    */
  printf("Aufruf von f3( )\n");
  i = k * j + x * f4();
  return i;
}

static int f4( )  {

  static int y;
  /* mit 0 init, behaehlt Wert */
  printf("Aufruf von f4( )\n");
  y = y + x;
}
```

Verarbeitung der

Unterprogramme



Verarbeitung des Hauptprogramm

`cpp`

Precompiler:

```
gcc -E Demo.c > Demo.i
```

`-E precompile`

`cc1`

Compiler:

```
gcc -S Demo.c
```

`-S assemble`

`as`

Assembler:

```
gcc -c Demo.c
```

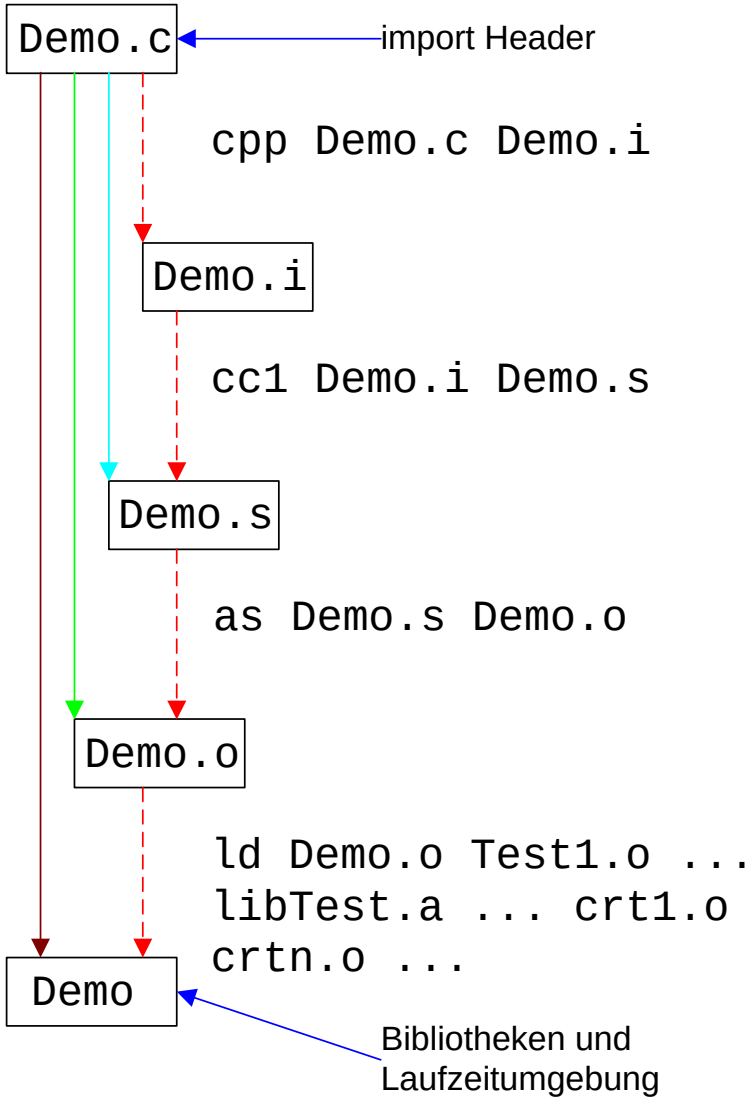
`-c compile`

`ld`

Linker/ Lader:

```
gcc Demo.c Test1.o -I.
-o Demo -L. -lTest
```

`-I includepath`
`-L librarypath`
`-l library`
`-o output file`



Ausgabe nach der Vorübersetzung (`Flag -E`)

```
# 1 "Demo.c"
# 1 "/usr/include/stdio.h" 1 3
# 1 "/usr/include/features.h" 1 3
extern int fprintf (FILE *__restrict __stream,
                    __const char *__restrict __format, ...);
extern int printf (__const char *__restrict __format, ...);
extern int sprintf (char *__restrict __s,
                    __const char *__restrict __format, ...);
...
extern int fscanf (FILE *__restrict __stream,
                   __const char *__restrict __format, ...);
extern int scanf (__const char *__restrict __format, ...);
# 1 "Test1.h" 1
extern int f1(void);
extern void f2(int *l);
# 1 "Test2.h" 1
extern int f3(int k);
extern int j;

main()  {
int i;
...
  printf( "Hauptprogramm\n");
...
  return 0;
}
```

Ausgabe nach der Übersetzung (`Flag -S`)

```
.file "Demo.c"
  .version  "01.01"
gcc2_compiled.:
.section  .rodata
.LC0:
  .string "Hauptprogramm\n"
.LC1:
  .string "i = %d\n"
.text
  .align 4
.globl main
  .type  main,@function
main:
  pushl %ebp
  movl %esp,%ebp
  subl $24,%esp
  addl $-12,%esp
  pushl $.LC0
  call printf
  addl $16,%esp
  call f1
  movl %eax,%eax
  movl %eax,-4(%ebp)
  addl $-12,%esp
  movl -4(%ebp),%eax
  pushl %eax
```

```
  call f3
  addl $16,%esp
  addl $-8,%esp
  movl -4(%ebp),%eax
  pushl %eax
  pushl $.LC1
  call printf
  addl $16,%esp
  xorl %eax,%eax
  jmp .L2
  .p2align 4,,7
.L2:
  movl %ebp,%esp
  popl %ebp
  ret
.Lfe1:
  .size  main,.Lfe1-main
  .ident  "GCC: (GNU) 2.95.3 19991030
           (prerelease)"
```

Parameter für den Vorübersetzer (`Flag -D`)

```
#ifdef C
#include <stdio.h>
#else
#ifdef CPP
#include <iostream>
#else
#error "Kein C / C++ Quelltext"
#endif
#endif

main( )  {
#ifdef C
  printf("Das ist die C Ausgabe\n");
#endif
#ifdef CPP
  cout << "Das ist die C++ Ausgabe" << endl;
#endif
  return 1;
}
```

```
(~/Demo) g++ C-CPP.cc -o C-CPP
         C-CPP.cc:7: #error "Kein C / C++ Quelltext"
(~/Demo) g++ -DC C-CPP.cc -o C-CPP && ./C-CPP
         Das ist die C Ausgabe
(~/Demo) g++ -DCPP C-CPP.cc -o C-CPP && ./C-CPP
         Das ist die C++ Ausgabe
```

Der vollständige Verarbeitungsprozess (Flag –v)

```
[root@localhost demo]# gcc -v Demo.c Test1.o -o Demo -L. -lTest
Reading specs from /usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/specs
gcc version 2.95.3 19991030 (prerelease)
/usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/cpp -lang-c -v -D__GNUC__=2 -D__GNUC_MINOR__=95 -
D__ELF__ -Dunix -D__i386__ -Dlinux -D__ELF__ -D__unix__ -D__i386__ -D__linux__ -D__unix -
D__linux -Asystem(posix) -Acpu(i386) -Amachine(i386) -Di386 -D__i386 -D__i386__ -Di586 -
Dpentium -D__i586 -D__i586__ -D__pentium -D__pentium__ Demo.c /tmp/ccn65Esd.i
GNU CPP version 2.95.3 19991030 (prerelease) (i386 Linux/ELF)
#include "..." search starts here:
#include <...> search starts here:
/usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/../../../../i586-mandrake-linux/include
/usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/include
/usr/include
End of search list.
The following default directories have been omitted from the search path:
/usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/../../../../include/g++-3
/usr/local/include
End of omitted list.
/usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/cc1 /tmp/ccn65Esd.i -quiet -dumpbase Demo.c -
version -o /tmp/cc2qlJ1h.s
GNU C version 2.95.3 19991030 (prerelease) (i586-mandrake-linux) compiled by GNU C version 2.95.3 19991030
(prerelease).
as -V -Qy -o /tmp/ccn1juAn.o /tmp/cc2qlJ1h.s
GNU assembler version 2.9.5 (i686-pc-linux-gnu) using BFD version 2.9.5.0.31
/usr/lib/gcc-lib/i586-mandrake-linux/2.95.3/collect2 -m elf_i386 -dynamic-linker /lib/ld-
linux.so.2 -o Demo /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/gcc-lib/i586-mandrake-
linux/2.95.3/crtbegin.o -L. -L/usr/lib/gcc-lib/i586-mandrake-linux/2.95.3 -L/usr/i586-
mandrake-linux/lib /tmp/ccn1juAn.o Test1.o -lTest -lgcc -lc -lgcc /usr/lib/gcc-lib/i586-
mandrake-linux/2.95.3/crtend.o /usr/lib/crtn.o
```

Optimierung

Anwenden:

```
gcc –[O | O2 | O3] Demo.c Test1.c Test2.c
```

`ohne O`	:	Reduzierung der Übersetzungzeit und Erzeugen eines Codes welcher für den Debugger sinnvoll ist.
`mit O`	:	Der Compiler versucht Code Grösse und Ausführungszeit zu reduzieren
`-O0`	:	Unterdrücken der Optimierung
`-O`	:	Optimierung zu Lasten der Übersetzungszeit
`-O1`	:	Optimierte Übersetzung
`-O2`	:	Optimierung (etwas mehr)
`-O3`	:	Optimierung (etwas mehr), inklusive Aktivierung von inline function
`-OX`	:	...
`-fflags`	:	weitere Optimierungsparameter (siehe `man gcc`)

Warnungen/Syntax

Anwenden:

```
gcc –[w | W] Demo.c Test1.c Test2.c
```

`-w`	:	Unterdrückt alle Warnungen
`-W`	:	Anzeigen von zusätzlichen Warnungen
`-pedantic`	:	Erzwingen von ANSI C
`-Wflags`	:	weitere Warnungsparameter (siehe `man gcc`)

Fehlermeldungen (Bsp. GNU Compiler und Linker)

Bsp. Fehlermeldung beim compilieren

1. Parse Error

```
1  #include <stdio.h>
2
3  main( )  {
4
5    printf("Hello world - compilererror\n")
6    return 1;                              ^
7  }                                        |
                                            |
g++     bsp02.cc   -o bsp02
bsp02.cc: In function `int main()':
bsp02.cc:6: parse error before "return"
gmake: *** [bsp02] Error 1
```

- Parse error verweist meistens auf ein fehlendes Semikolon oder eine fehlerhafte Klammerung der Blockstruktur `{ }`.
- Fehlende schliessende Klammer
  `parse error at end of input`

2. Missing ...

```
1  #include <stdio.h>
2
3  main( )  {
4
5    printf("Hello world - compilererror\n);
6    return 1;                                  ^
7  }                                            |
                                                |
g++     bsp02.cc   -o bsp02
bsp02.cc:5:16: missing terminating " character
gmake: *** [bsp02] Error 1
```

3. Klammerfehler ( )

```
1  #include <stdio.h>
2  extern void f1( );
3
4  main( ) {
5
6      int i;
7      if i > 0
8          printf("Hello world - error\n");
9      f1;
10     return 1;
11  }
```

// g++ egcs-2.91.66 19990314
"bsp02.cc", line 7: Error: "(" expected instead of "i".

// g++ 2.97 20001016 (experimental)
bsp02.cc:7: parse error before '>' token
bsp02.cc:9: warning: statement is a reference, not call, to function `f1'
bsp02.cc:11: Internal error #122.
bsp02.cc:11: Internal compiler error in , at ../gcc/cp/decl.c:14232

- Ausdrücke in Steuerkonstrukte müssen geklammert werden.
- Funktionen/Prozeduren müssen Klammern haben, auch wenn sie parameterlos sind.

4. Undefined Symbol

```
1  #include <stdio.h>
2  extern void f1( int);
3
4  main( ) {
5
6      printf("Hello world - linkererror\n");
7      f1( 1);
8      return 1;
9  }
```

```
g++     bsp02.cc   -o bsp02
Undefined                first referenced
symbol                   in file
f1(int)                  /var/tmp/ccyDu6aO.o
ld: fatal: Symbol referencing errors. No output written to bsp02
collect2: ld returned 1 exit status
gmake: *** [bsp02] Error 1
```

- Referenzieren einer Prozedur in einem anderen Modul.
- Ein Modul mit dieser Prozedur wird aber nicht beim Verbinden angegeben.
- Wenn die Datei f1.cc die Prozedur `void f1(int)` enthält, könnte es wie folgt aussehen.

Übersetzen des separaten Modules:

```
g++ -c f1.cc
```

Ergibt: `f1.o`

Übersetzen des Hauptprogramms mit Anbinden des Moduls

```
g++ bsp02.cc f1.o -o bsp02
```

5. Symbol multiply defined

```
1  #include <stdio.h>
2
3  main( )  {
4
5    printf("Hello world - linkererror\n");
6    return 1;
7  }
```

- Die Moduln bsp02.cc und bsp0x.cc enthalten eine Funktion mit gleichem Namen (Signatur).

```
g++ bsp02.cc bsp0x.cc -o bsp02
ld: fatal: symbol `main' is multiply defined:
          (file /var/tmp/ccnmNNag.o and file
                /var/tmp/cc0wL6Ah.o);
ld: fatal: File processing errors.
           No output written to bsp02
collect2: ld returned 1 exit status
```

- Temporäre Dateien z.B. **`/var/tmp/ccnmNNag.o`** entstehen während des Übersetzungprozesses zur Übergabe des Codes zwischen den Phasen der Programmerzeugung. (Wichtig: freier Speicherplatz in `tmp`)

Fehlersuche II (Laufzeitfehler)

unterschiedliche Möglichkeiten (wichtig systematisch)

1. Eingrenzung des Fehlers

   - Auskommentierung von Quelltextbereichen

   - Einfügen von Ausgabenanweisungen (Statusmeldungen) über die ungepufferte Ausgabe

   C - `fprintf` to `stderr`, C++ - `cerr`

```
double *ptr = NULL;
fprintf(stderr, "before error code\n");
scanf("%f", ptr);                          // write into a NULL pointer
cerr << "after error code" << endl;
```

Ausgabe:

```
before error code
43
Segmentation fault (core dumped)
```

Anmerkung:

```
limit core 0  # in .[t]cshrc verhindert das Schreiben des core dumps
```

2. Reaktion auf die Ausführung des fehlerhaften Codes

- Verwendung von assert (z.B. bei Zeigern)

```
#include <assert.h>
...
double *ptr = NULL;
assert(ptr);
scanf("%f", ptr);   // write into a NULL pointer
```

Ausgabe:

```
runerror.cc:13: failed assertion `ptr'
Abort (core dumped)
```

- Benutzung von Debuggern zur Schrittweise Abarbeitung des Programms (z.B.)

```
GDB 7.3.50  (cygwin/Solaris 11)    // Kommandozeile
DDD 3.3.12  (cygwin)               // Uni Braunschweig
```

Vorraussetzung: Übersetzen des Programmes mit debug Option

```
gcc/g++   -g <datei>.c -o <datei>
           ^
           |
       Compiler erzeugt symbolische Informationen zum debuggen.
```

Beispiel mit Fehlern:

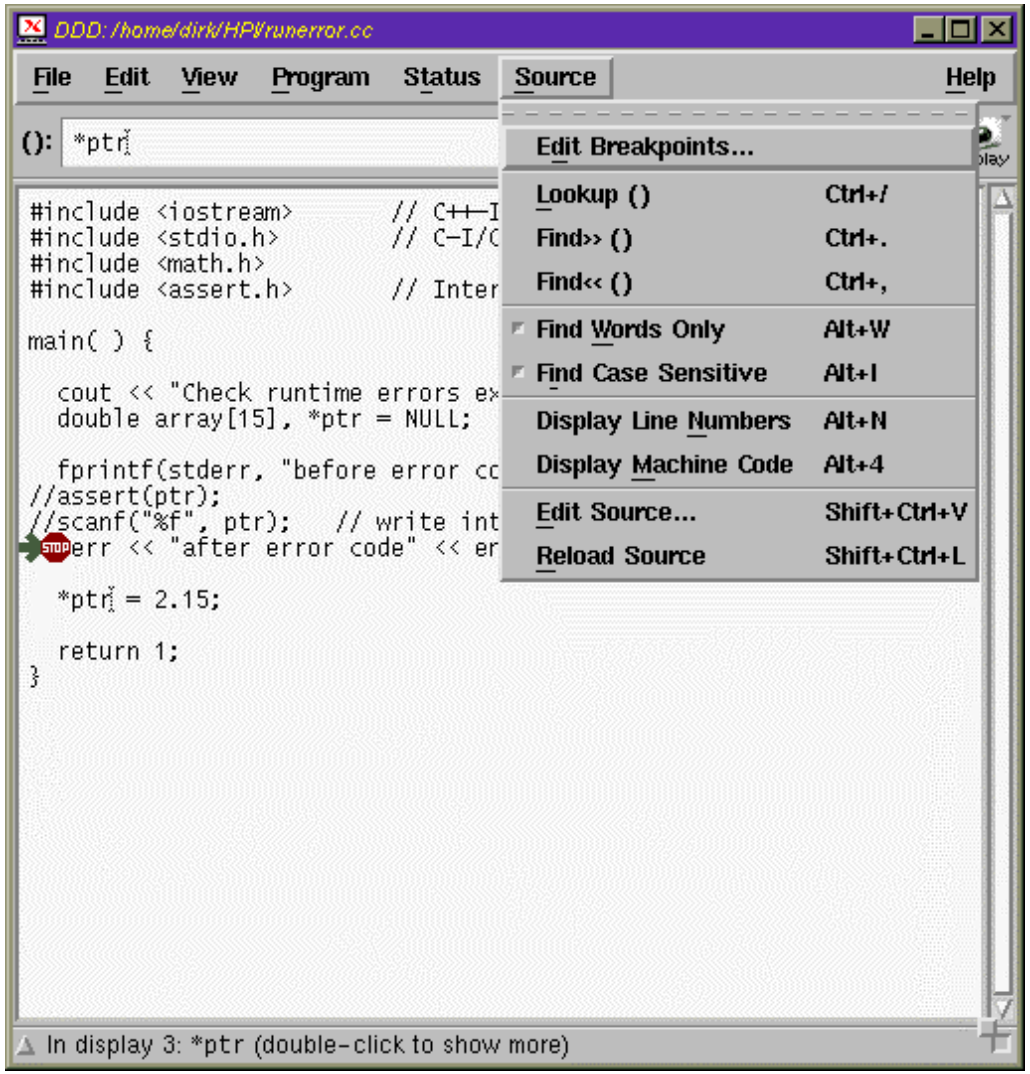
```
#include <stdio.h>          // C-I/O Interface
#include <math.h>
#include <assert.h>         // Interface to assert function

main( ) {

  printf("Check runtime errors example\n");

  double array[15], *ptr = NULL;

  fprintf(stderr, "before error code\n");
//assert(ptr);
//scanf("%f", ptr);   // write into a NULL pointer
  printf(“after error code\n");

  *ptr = 2.15;

  return 1;
}
```

gdb

Starten

```
gdb runerror       // Name der ausführbaren Datei

(gdb) run
   Starting program: /home/dirk/HPI/runerror
   Check runtime errors example
   ...
   Program received signal SIGSEGV, Segmentation fault.
   0x10a00 in main () at runerror.cc:16
   16        *ptr = 2.15;
(gdb) break 16
   Breakpoint 1 at 0x109f4: file runerror.cc, line 16.
(gdb) run
   The program being debugged has been started already.
   Start it from the beginning? (y or n) y
   ...
   Breakpoint 1, main () at runerror.cc:16
   16        *ptr = 2.15;
(gdb) print ptr
   $1 = (double *) 0x0
(gdb) quit
   The program is running.  Quit anyway (and kill it)? (y or n) y
```

ddd

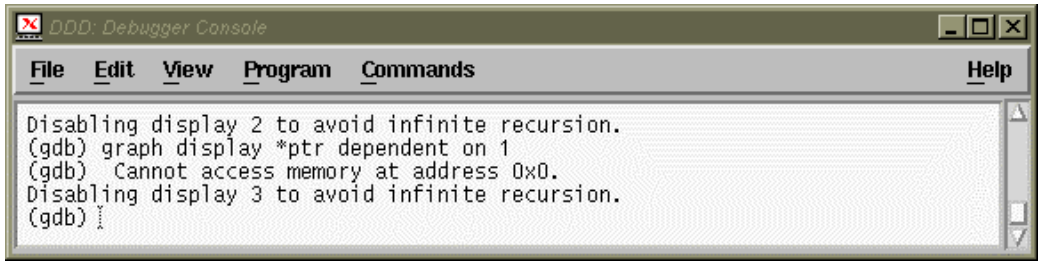


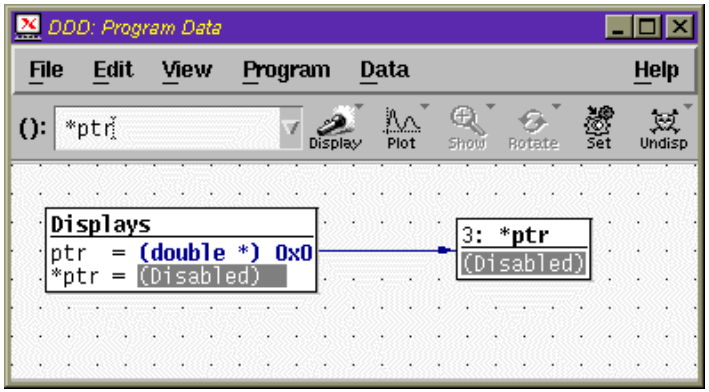
Starten

```
ddd runerror //Name der ausführbaren Datei
```

Source Window

Setzen eines Haltepunktes über Kontextmenü


DDD: Debugger Console
File Edit View Program Commands Help
Disabling display 2 to avoid infinite recursion.
(gdb) graph display *ptr dependent on 1
(gdb) Cannot access memory at address 0x0.
Disabling display 3 to avoid infinite recursion.
(gdb)


DDD: Program Data
File Edit View Program Data Help
(): *ptr
Display Plot Show Rotate Set Undisp
Displays
ptr = (double *) 0x0
*ptr = (Disabled)
3: *ptr
(Disabled)


DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make