Tools

“to get a deeper understanding of the language”

Deep C - a 3 day course

Jon Jagger & Olve Maudal

A glimpse into tools often used when developing C

Exercise: Deep thought, Part 1

dt.c

```c
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

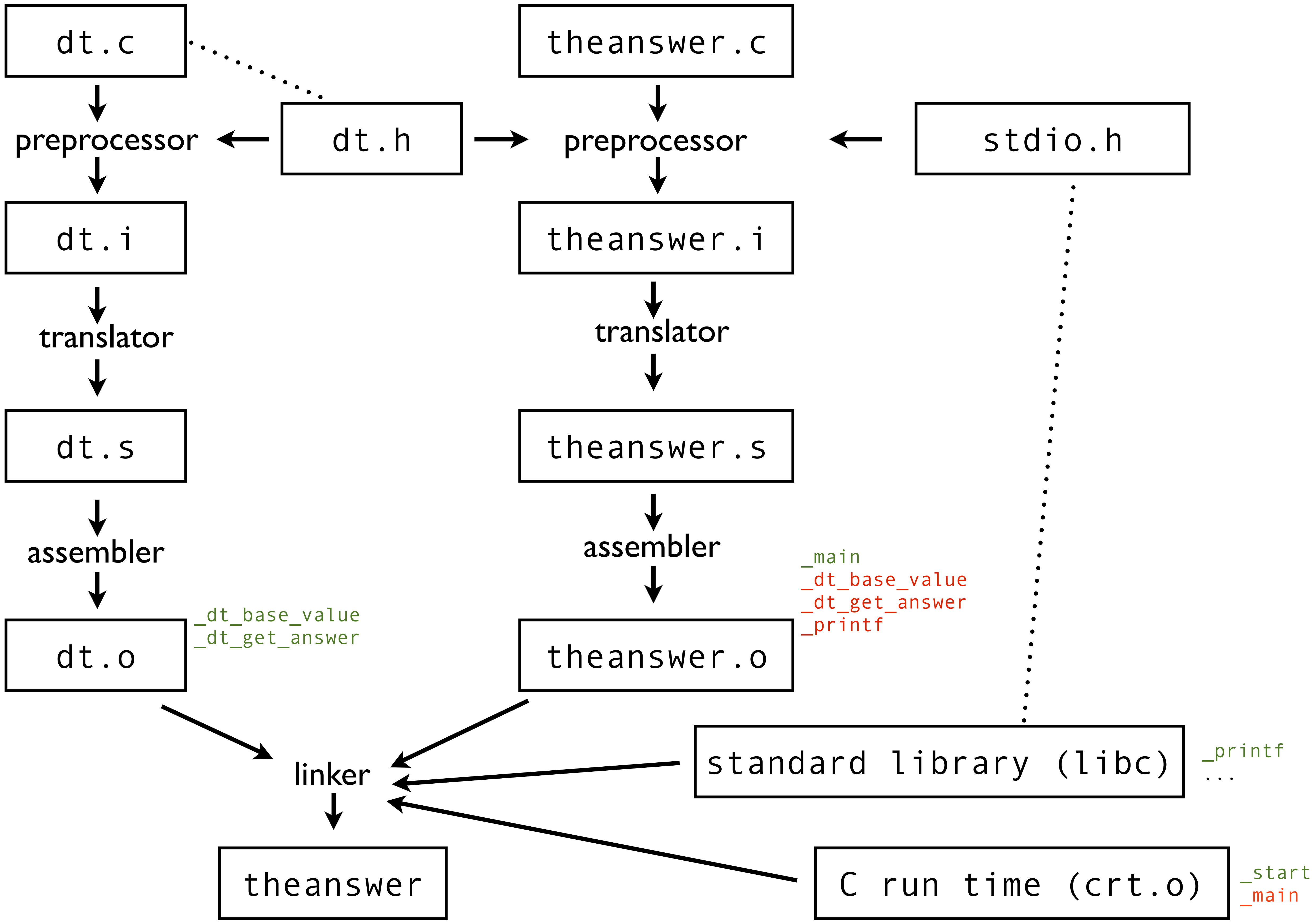
```c
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

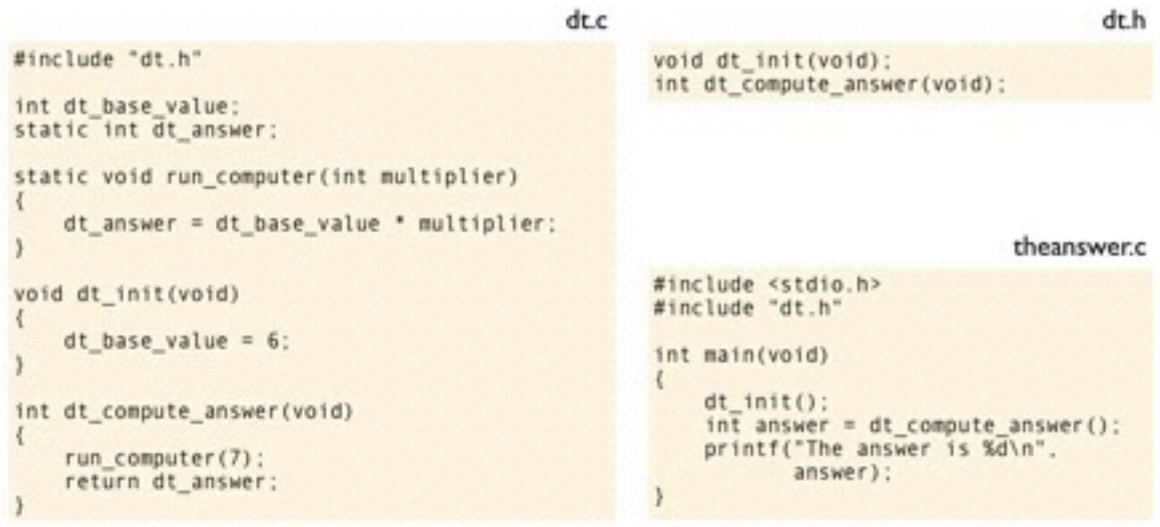
```c
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -c dt.c
$ cc -c theanswer.c
$ cc -o theanswer theanswer.o dt.o
$ ./theanswer
The answer is 42
$
```


dt.c
theanswer.c
preprocessor
dt.h
preprocessor
stdio.h
dt.i
theanswer.i
translator
translator
dt.s
theanswer.s
assembler
assembler
_main
_dt_base_value
_dt_get_answer
_printf
dt.o
_dt_base_value
_dt_get_answer
theanswer.o
linker
standard library (libc)
_printf
...
theanswer
C run time (crt.o)
_start
_main

Exercise: Deep thought, Part 2



dt.c

```
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}
```

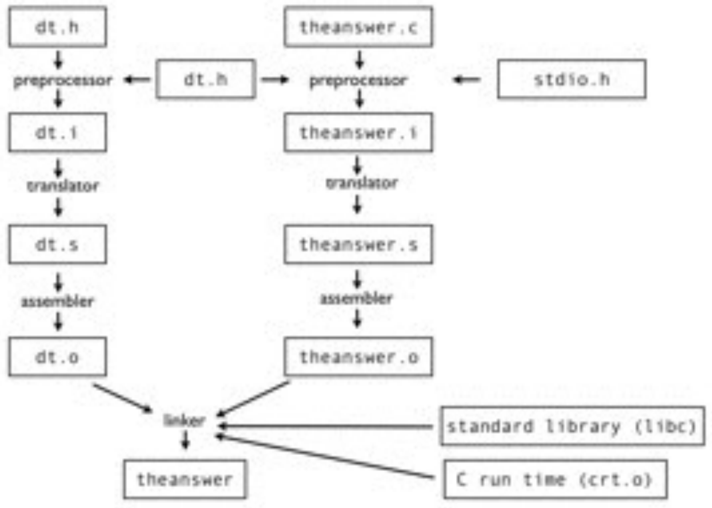
dt.h

```
void dt_init(void);
int dt_compute_answer(void);
```

theanswer.c

```
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
           answer);
}
```

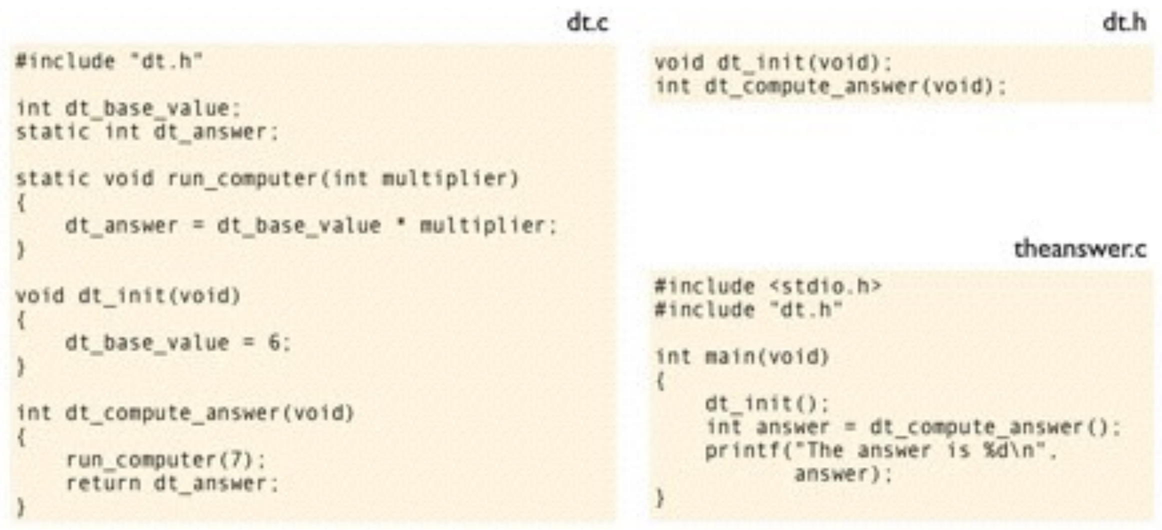


```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o

$ cc -c -save-temps theanswer.c
$ ls theanswer.*
$ nm theanswer.o

$ ld -lc -o theanswer dt.o theanswer.o /usr/lib/crt1.o
$ ./theanswer
The answer is 42
```

Exercise: Deep thought, Part 3



dt.c

```
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}
```

dt.h

```
void dt_init(void);
int dt_compute_answer(void);
```

theanswer.c

```
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break run_computer
(gdb) run
(gdb) set dt_base_value = 8
(gdb) cont
(gdb) disassemble run_computer
(gdb) set disassembly-flavor intel
(gdb) disassemble run_computer
(gdb) help
(gdb) quit
```

test seams

- suppose I want to test a scenario where getaddrinfo() fails

message.c

```
#include "message.h"
#include <netdb.h>  // getaddrinfo();
...

int send_message(const char * msg, size_t size)
{
    struct addrinfo hints =
    {
        .ai_family = AF_UNSPEC,
        .ai_socktype = SOCK_STREAM,
        .ai_flags = AI_PASSIVE
    };
    const char * port = "3490";
    struct addrinfo * serv_info;
    int rv = getaddrinfo(NULL, port, &hints, &serv_info);
    if (rv != 0)
    {
        fputs("getaddrinfo: ", stderr);
        fputs(gai_strerror(rv), stderr);
        fputc('\n', stderr);
        return EXIT_FAILURE;
    }
    ...
}
```

include test seam

- [-iquote folder] adds a new #include "..." folder
- [-isystem folder] adds a new #include "..."/<...> folder

```
CFLAGS += -iquote  ./local_seam
CFLAGS += -isystem ./system_seam

target: ...
     @gcc -v $(CFLAGS) ... -o $@
```

makefile

verbose

```
#include "..." search starts here:
 ./local_seam
#include <...> search starts here:
 ./system_seam
 /usr/lib/gcc/x86_64-linux-gnu/4.8/include
 /usr/local/include
 /usr/lib/gcc/x86_64-linux-gnu/4.8/include-fixed
 /usr/include/x86_64-linux-gnu
 /usr/include
End of search list.
```

fake function test seam

- a test can be compiled against selected fake functions

test_send_message/getaddrinfo_failure_prints_gai_diagnostic_to_stderr.c

```
#include "message.h"
#include <netdb.h>
...
int getaddrinfo(const char * host_name, const char * server_name,
                const struct addrinfo * hints, struct addrinfo ** result)
{
    ...
    return 42; // force failure
}                                        ←

int main(void)
{
    const char message[] ="Hello, world";
    int rv = send_message(message, sizeof message);
    assert(rv == EXIT_FAILURE);
    assert(...);
    assert(...);
}
```

message.c

```
...
```

fake function test seam

- the excellent FFF* makes writing fakes very easy and allows multiple tests per source file

test_send_message.c

```
#include "message.h"
#include "fff.h"
...
FAKE_VALUE_FUNC(int, getaddrinfo, const char *, const char *,
                const struct addrinfo *, struct addrinfo **)
FAKE_VALUE_FUNC(int, fputs, const char *, FILE *)
...
static void getaddrinfo_failure_prints_gai_diagnostic_to_stderr(void)
{   ...
    getaddrinfo_fake.return_val = EAI_AGAIN;

    const char message[] ="Hello, world";
    int rv = send_message(message, sizeof message);
    assert(rv == EXIT_FAILURE);
    assert(getaddrinfo_fake.call_count == 1);
    assert(fputs_fake.call_count == 2);
    assert(strcmp("getaddrinfo: ", fputs_fake.arg0_history[0]) == 0);
    assert(fputs_fake.arg1_history[0] == stderr);
    assert(strcmp(gai_strerror(getaddrinfo_fake.return_val),
                  fputs_fake.arg0_history[1]) == 0);
    assert(fputs_fake.arg1_history[1] == stderr);
    ...
}
```

*Fake Function Framework: https://github.com/meekrosoft/fff

link time test seam

- use [-Wl,--wrap=func] to redirect func to __wrap_func
- only works *across* translation units

```
target: dependencies
     @gcc -Wl,--wrap=getaddrinfo ... -o $@
```

makefile

test_send_message/getaddrinfo_failure_prints_gai_diagnostic_to_stderr.c

```
#include "message.h"
...
int __wrap_getaddrinfo(const char * host_name, const char * server_name,
                       const struct addrinfo * hints, struct addrinfo ** result)
{
    ...
    return 42;
}

int main(void)
{
    const char message[] ="Hello, world";
    int rv = send_message(message, sizeof message);
    assert(rv == EXIT_FAILURE);
    assert(...);
    assert(...);
}
```

run time test seam

- using function pointers

```
#include "..."

struct netdb_api
{
  int (*getaddrinfo)(const char *, const char *,
                     const struct addrinfo *, struct addrinfo **);
    ...
};

extern const struct netdb_api netdb;
```

```
#include "message.h"
#include "netdb_api.h"

int send_message(const char * msg, size_t size)
{
    ...
    const char * port = "3490";
    struct addrinfo * serv_info;
    int rv = netdb.getaddrinfo(NULL, port, &hints, &serv_info);
    if (rv != 0)
    {
        fputs("getaddrinfo: ", stderr);
        fputs(gai_strerror(rv), stderr);
        fputc('\n', stderr);
        return EXIT_FAILURE;
    }
    ...
}
```

run time test seam

- using function pointers
  also allows multiple tests per source file

```
#include "message.h"
...
#define UNUSED(var)  ((void)var)

int fake_getaddrinfo(
    const char * host_name,
    const char * server_name,
    const struct addrinfo * hints,
    struct addrinfo ** result)
{
    UNUSED(host_name);
    ...
    return 42; // force failure
}

static void getaddrinfo_failure_prints_gai_diagnostic_to_stderr(void)
{
    const char message[] ="Hello, world";
    netdb.getaddrinfo = fake_getaddrinfo;
    int rv = send_message(message, sizeof message);
    assert(rv == EXIT_FAILURE);
    assert(...);
    assert(...);
}
```

summary

- hello world!
- behaviour
- vocabulary of the language
- preprocessor, translator, assembler, linker
- standard library and C run-time
- test seams
  - extra #include paths
  - fake functions
  - linker wrapping
  - function pointers