Skipulag forrita í C, make, gdb

Tölvunarfræði 2, vor 2012

Hallgrímur H. Gunnarsson

Háskóli Íslands

2012-01-20

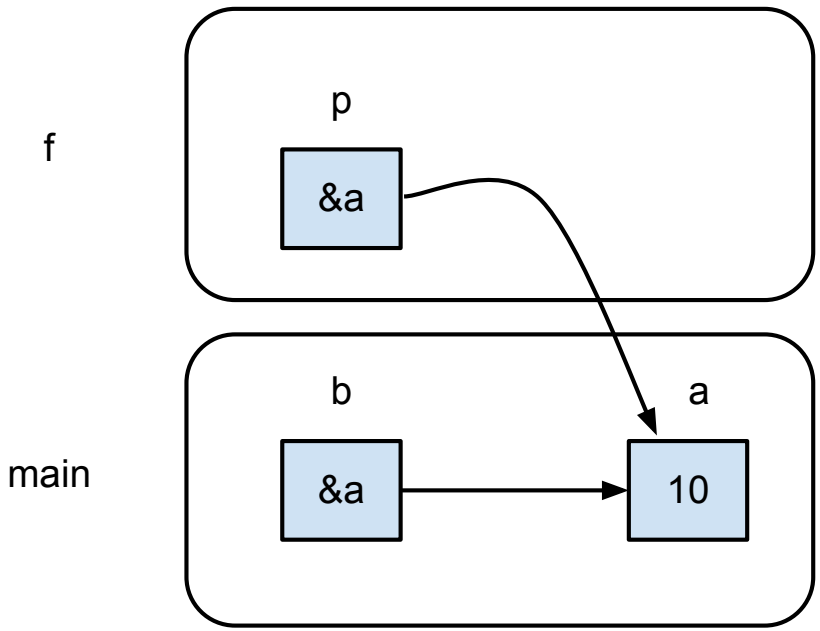
Bendar

```c
void f(int *p) {
  *p = 20;
}

int main(void) {
  int a = 10;
  int *b = &a;

  f(&a);
  // eða jafngilt
  f(b);

  printf("%d\n", a);
}
```



Þýðing og keyrsla

```
hhg@hhg:~/t2$ gcc -ggdb3 q.c -o q
hhg@hhg:~/t2$ gdb ./q
GNU gdb (Ubuntu/Linaro 7.2-1ubuntu11) 7.2
[..]
Reading symbols from /home/hhg/t2/q...done.
(gdb) break main
Breakpoint 1 at 0x80483db: file q.c, line 9.
(gdb) run
Starting program: /home/hhg/t2/q

Breakpoint 1, main () at q.c:9
9 int a = 10;
(gdb) where
#0  main () at q.c:10
(gdb)
```

Höldum áfram

```
(gdb) info locals
a = 10
b = 0xbffff148
(gdb) p &a
$1 = (int *) 0xbffff148
(gdb) step
12 f(&a);
(gdb) step
f (p=0xbffff148) at q.c:5
5 *p = 20;
(gdb) bt
#0  f (p=0xbffff148) at q.c:5
#1  0x080483f7 in main () at q.c:12
(gdb) info args
p = 0xbffff148
(gdb)
```

gdb

Keyrsla:

- run - Ræsir forritið
- kill - Drepur forritið
- quit - Hættir í gdb

Breakpoints:

- break <function> – Stöðvar keyrslu við innkomu í gefið fall
- break <linenr> – Stöðvar keyrslu í gefnu línunúmeri
- break <filename:linenr> – Stöðvar keyrslu á gefnum stað

Forritstexti:

- where – Sýnir núverandi staðsetningu
- list – Sýnir forritstexta í kringum núverandi staðsetningu

gdb

Framvinda:

- step – Framkvæmir næstu línu, fer inn í föll
- next – Framkvæmir næstu línu, fer ekki inn í föll
- continue – Framkvæmir fram að næsta breakpoint
- finish – Framkvæmir restina af núverandi falli

Vakningarfærslur á hlaða:

- bt – Sýnir stack trace
- bt full – Stack trace ásamt staðværum breytum
- down – Ferðast niður í næstu vakningarfærslu
- up – Ferðast upp í fyrri vakningarfærslu
- info locals – Sýnir staðværar breytur
- info args – Sýnir viðföng
- info frame – Sýnir núverandi vakningarfærslu

gdb

Print:

- p <expr> – Prentar segð
- p/x <expr> – Prentar segð sem hex
- p/s <expr> – Prentar segð sem streng
- p/d <expr> – Prentar segð sem tölu
- p/t <expr> – Prentar segð sem tvítölu (binary)

Examine:

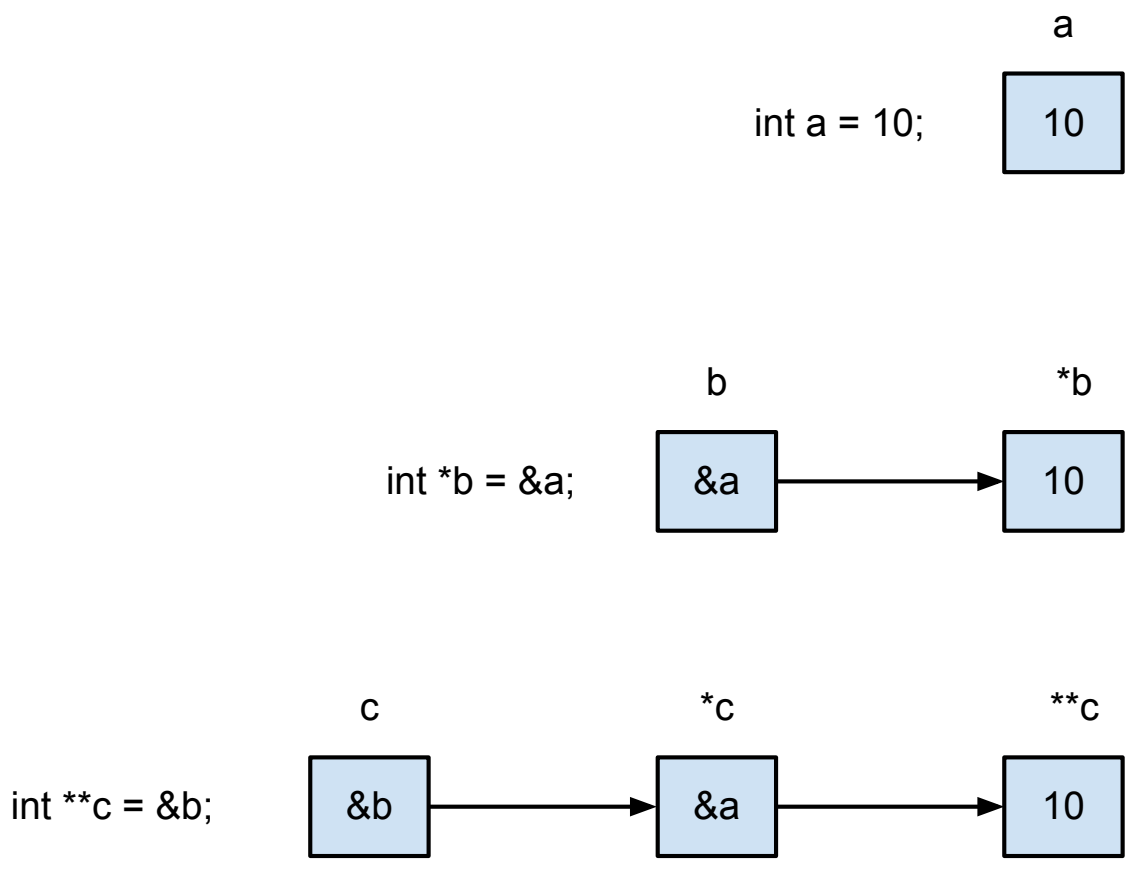
- x <address> – Sýnir gildi í vistfangi
- x/x <address> – Sýnir gildi í vistfangi sem hex
- x/s <address> – sem streng
- x/d <address> – sem tölu
- x/t <address> – sem tvítölu

Hreiðraðir bendar

```c
int main(void)
{
  int a = 10;
  int *b = &a;
  int **c = &b;

  printf("%d\n", a);
  printf("%d\n", *b);
  printf("%d\n", **c);
}
```

```
hhg@hhg:~/t2$ ./w
10
10
10
hhg@hhg:~/t2$
```

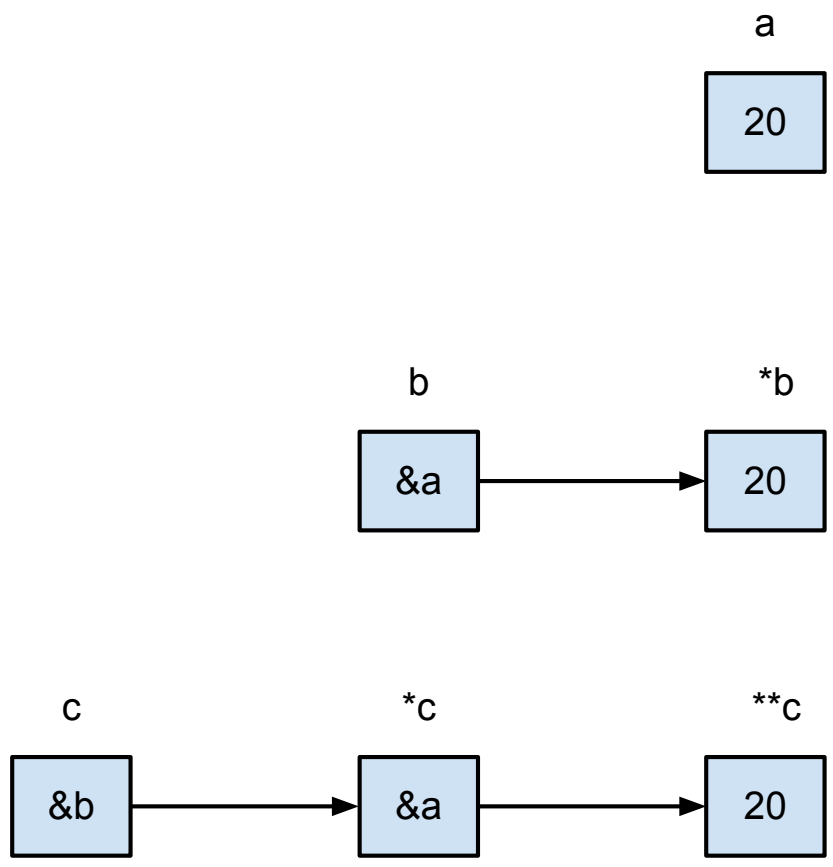


Hreiðraðir bendar

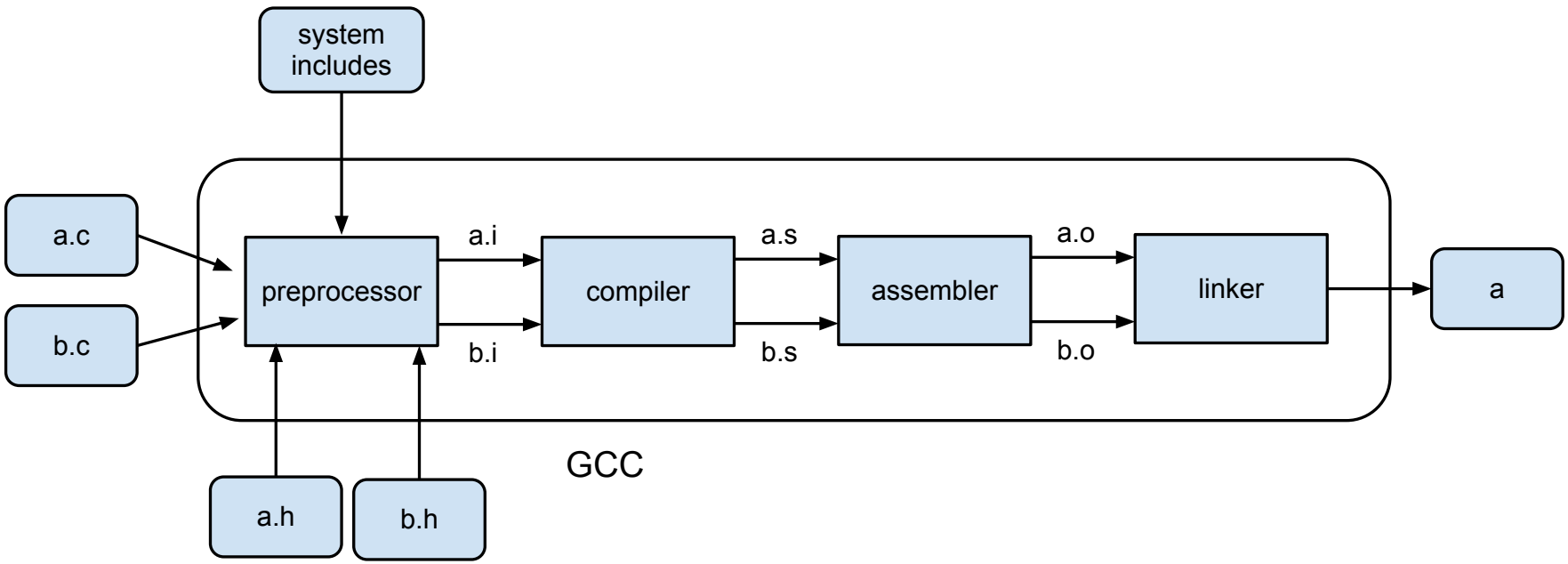
```c
int main(void)
{
  int a = 10;
  int *b = &a;
  int **c = &b;

  a = 20;
  *b = 20;
  **c = 20;

  printf("%d\n", a);
  printf("%d\n", *b);
  printf("%d\n", **c);
}
```



GCC


system
includes
a.c
b.c
preprocessor
a.i
b.i
compiler
a.s
b.s
assembler
a.o
b.o
linker
a
GCC
a.h
b.h

Declaration vs. definition

Breytur:

- Declaration: s og k breyturnar eru til, en þær eru skilgreindar á öðrum stað

```
// file1.c - declare
extern char s[10];
extern int k;
```

- Definition: Skilgreining á s og k sem tekur frá minni fyrir þær

```
// file2.c - define
char s[10];
int k;
```

Föll:

- Declaration: f er til, en það er skilgreint á öðrum stað

```
// Declare
int f(int a, int b);
```

- Definition: Raunveruleg skilgreining á f

```
// Define
int f(int a, int b)
{
}
```

Header skrár

Notkun á header skrám:

- Sameiginlegar skilgreiningar á structs
- Skilgreining á föstum, macros, o.s.frv.
- Lýsing á föllum (e. function prototypes)

ATH. Komum í veg fyrir tvöfaldan include árekstur með `#include` með `#ifndef` og `#define`

```
#ifndef _TEST_H
#define _TEST_H

// ...

#endif
```

Dæmi um header skrá

```
#ifndef _STACK_H
#define _STACK_H

struct item {
  int value;
  struct item *next;
};

struct stack {
  struct item *top;
  int n;
};

extern void stack_init(struct stack *s);
extern void stack_push(struct stack *s, int value);
extern int stack_pop(struct stack *s);
extern int stack_empty(struct stack *s);

#endif
```

stack forrit

```c
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

void stack_init(struct stack *s)
{
  s->top = NULL;
  s->n = 0;
}

int stack_empty(struct stack *s)
{
  return s->top == NULL;
}

void stack_push(struct stack *s, int value)
{
  struct item *p = malloc(sizeof(struct item));

  p->value = value;
  p->next = s->top;
  s->top = p;
}
```

```c
int stack_pop(struct stack *s)
{
  struct item *p = s->top;
  int value = p->value;

  s->top = s->top->next;
  s->n--;

  free(p);

  return value;
}
```

main forrit

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

int main(int argc, char *argv[])
{
  struct stack s;

  stack_init(&s);
  stack_push(&s, 10);
  stack_push(&s, 20);
  stack_push(&s, 30);

  while (!stack_empty(&s))
  {
    printf("%d\n", stack_pop(&s));
  }
}
```

Þýðing

Skrárnar þýddar og tengdar báðar saman í einni skipun:

```
hhg@hhg:~/t2/stack$ gcc stack.c test.c -o test
hhg@hhg:~/t2/stack$ ./test
30
20
10
hhg@hhg:~/t2/stack$
```

Skrárnar þýddar sitt í hvoru lagi og síðan tengdar:

```
hhg@hhg:~/t2/stack$ gcc -c stack.c
hhg@hhg:~/t2/stack$ gcc -c test.c
hhg@hhg:~/t2/stack$ gcc stack.o test.o -o test
hhg@hhg:~/t2/stack$
```

Symbols

```
hhg@hhg:~/t2/stack$ nm test.o
00000000 T main
         U printf
         U stack_empty
         U stack_init
         U stack_pop
         U stack_push
hhg@hhg:~/t2/stack$ nm stack.o
         U free
         U malloc
0000008c T stack_empty
00000000 T stack_init
0000004a T stack_pop
00000018 T stack_push
hhg@hhg:~/t2/stack$
```

- U = undefined
- T = in text (code) section
- Lágstafur = local fyrir þessa object skrá
- Hástafur = global (external)

Symbols

Hvað gerist ef við reynum að tengja object skrár og það vantar symbols?

```
hhg@hhg:~/t2/stack$ gcc test.c -o test
/tmp/ccQxec2e.o: In function `main':
test.c:(.text+0x11): undefined reference to `stack_init'
test.c:(.text+0x25): undefined reference to `stack_push'
test.c:(.text+0x39): undefined reference to `stack_push'
test.c:(.text+0x4d): undefined reference to `stack_push'
test.c:(.text+0x5b): undefined reference to `stack_pop'
test.c:(.text+0x78): undefined reference to `stack_empty'
collect2: ld returned 1 exit status
hhg@hhg:~/t2/stack$
```

Makefile

```
CC = gcc
OBJS = stack.o test.o

all: test

test: $(OBJS)
        $(CC) $(OBJS) -o test

stack.o: stack.h
test.o: stack.h

.c.o:
        $(CC) $(CFLAGS) -c $*.c
```