Kynning á C

Tölvunarfræði 2, vor 2012

Hallgrímur H. Gunnarsson

Háskóli Íslands

2012-01-11

PDP-7

- 1969: Ken Thompson skrifar upphaflega UNIX stýrikerfið í smalamáli fyrir PDP-7 tölvurnar
- Vildi high-level forritunarmál til að halda þróun UNIX áfram
- Býr til nýtt forritunarmál sem hann kallar B
- B er byggt á BCPL ("stripped down BCPL"), sem er byggt á Algol

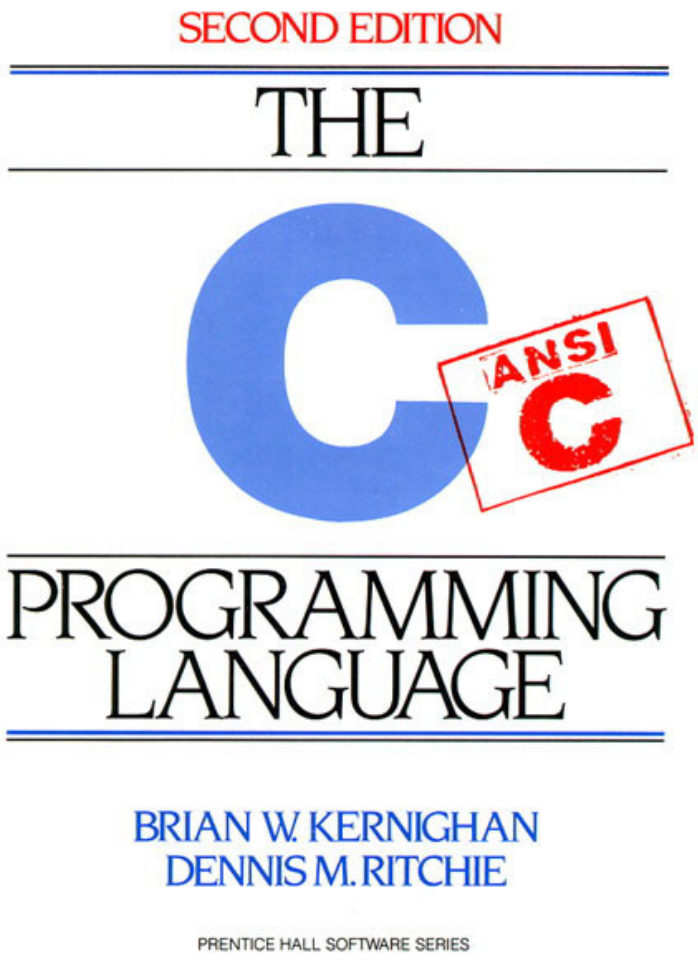
PDP-11

- 1970: UNIX hópurinn fær PDP-11 tölvu og flytur UNIX og B yfir á hana
- 1971: B reynist henta illa fyrir PDP-11
- 1971-1973: Dennis Ritchie býr til NB ("New B"), það endar sem C
- 1973: UNIX, þ.m.t. kjarni stýrikerfisins, er endurskrifaður í C

PDP-11

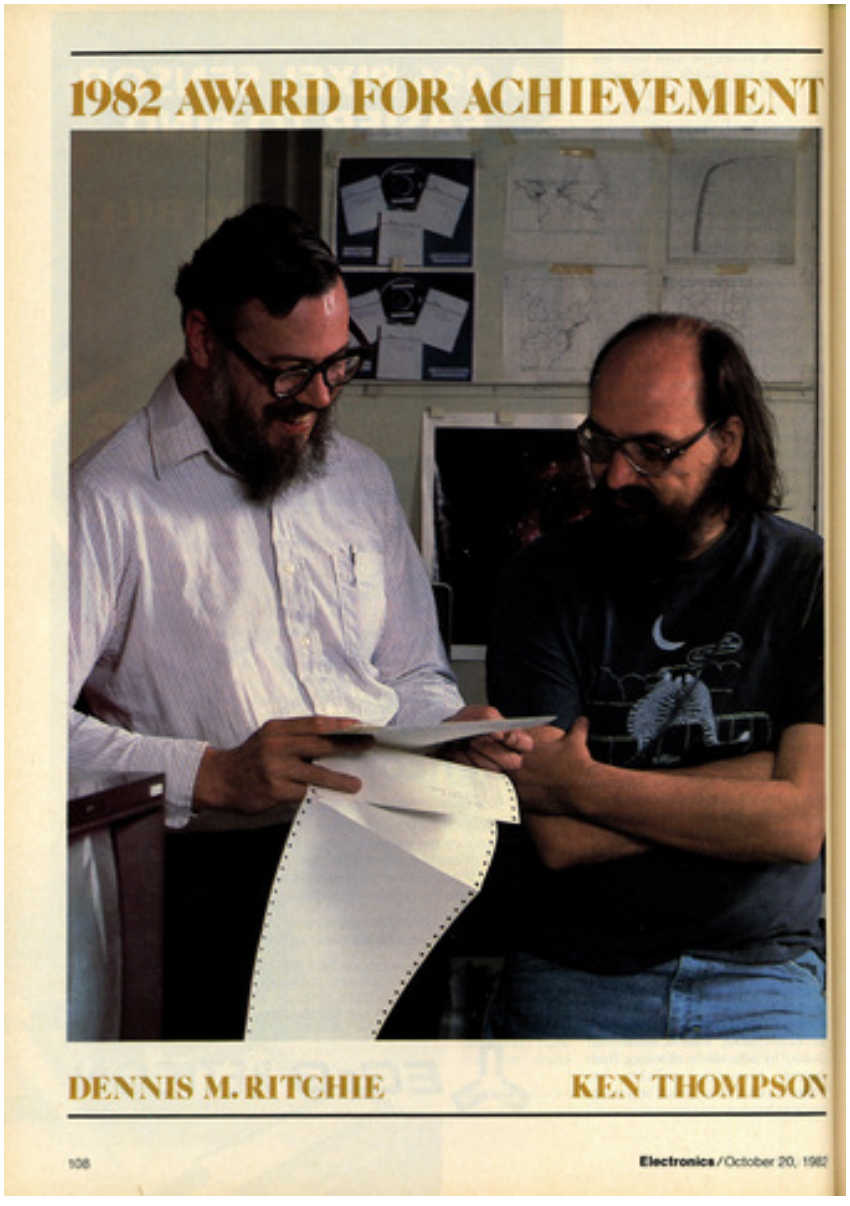
- Auðveldur flutningur (portability) var einn af stórum kostum þess að hafa UNIX í C
- Nægði að búa til C þýðanda fyrir nýja tölvu til að keyra UNIX á henni
- Óþarfi að skrifa allt upp á nýtt í nýju smalamáli þegar ný tölva kom út
- UNIX var eitt af fyrstu stýrikerfunum sem voru ekki skrifuð í smalamáli

K&R C



- C hélt áfram að þróast á áttunda áratugnum
- Árið 1978 kom út fyrsta bókin um C eftir Ritchie og Kernighan
- Enginn staðall til fyrir C þegar hún kom út og bókin varð de-facto staðall (þekktur sem K&R C)
- Til að eyða öllum vafa og koma í veg fyrir ósamhæfðar útgáfur af C þá var orðið nauðsynlegt að staðla C

Útgáfur af C



C staðlar:

- 1978: K&R C
- 1989: C89 (ANSI C)
- 1999: C99 (ISO C99)
- 2011: C11 (ISO C11)

Nýjungar í hverri útgáfu:

- C89: void pointers, function prototypes
- C99: variable-length arrays, inline functions, complex datatype, // comments
- C11: atomic operations, multi-threading, betri Unicode stuðningur

Kostir C

Kostir:

- C er lítið og einfalt forritunarmál
- C er low-level forritunarmál, hentar vel í kerfisforritun
- C er auðveldlega flytjanlegt (portable)
- C er hraðvirkt og vinnur nálægt járninu (sumir kalla það "portable assembler")
- C er statically typed (tög eru athuguð við þýðingu) sem útilokar margar villur

Dæmi um hugbúnað í C:

- Stýrikerfi:
  - UNIX
  - Linux
  - *BSD
- Forritunarmál:
  - Sun Java
  - CPython
  - Perl
  - PHP
  - GCC
- Leikjavélar:
  - Doom, Doom II
  - Quake II, Quake III

Ókostir C

Ókostir:

- **Weak typing**: C er statically typed, en tögunin í því er einnig mjög slök og leyfir hluti sem t.d. tögunin í Java bannar
- **Beittur hnífur**: C er mjög öflugt, en það getur líka verið auðvelt að gera mistök, lítil mistök geta haft mikil áhrif
- **Low-level**: C er low-level og nálægt vélinni sjálfri, ekki rétt stig af abstraction fyrir mörg forrit, viljum vinna nær vandamálinu sem er fyrir höndum, frekar en vélinni sjálfri
- **Ruslasöfnun**: C er ekki með ruslasöfnun, en flest high-level nútíma forritunarmál hafa ruslasöfnun
- **Einingaforritun**: C er með takmarkaðan stuðning fyrir einingaforritun og það getur verið erfitt að viðhalda stórum forritum í C

Fyrsta C forritið

```c
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
}
```

Þýðing

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

Keyrsla

```
hhg@hhg:~/t2$ ./hello
Hello world!
hhg@hhg:~/t2$
```

Fyrsta C forritið

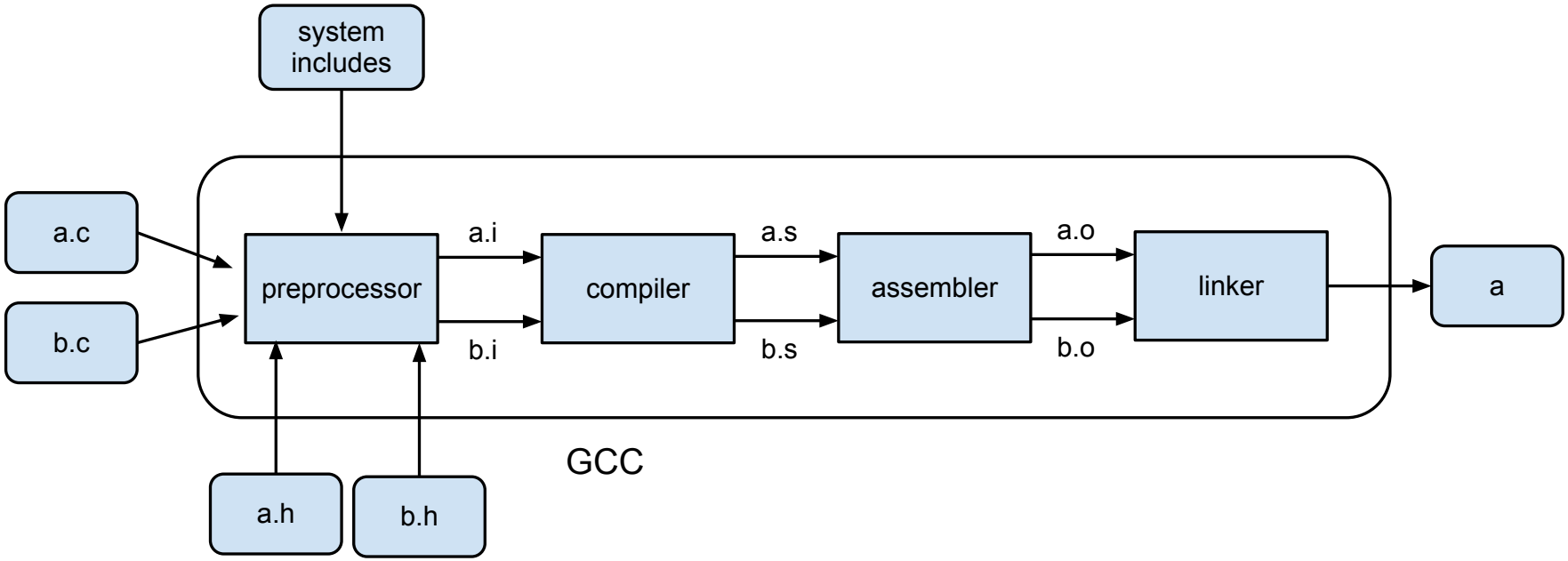
```c
#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
}
```

Útskýringar

- Föll í C standa sjálfstætt og þurfa ekki að vera hluti af klasa eins og í Java
- Allt sem byrjar á # er ætlað forþýðandanum (preprocessor), meira um það síðar
- Sjálfstætt keyranlegt forrit verður að innihalda main fall
- main fallið þjónar sem inngangspunktur forritsins, þegar það er ræst þá byrjar keyrslan í main

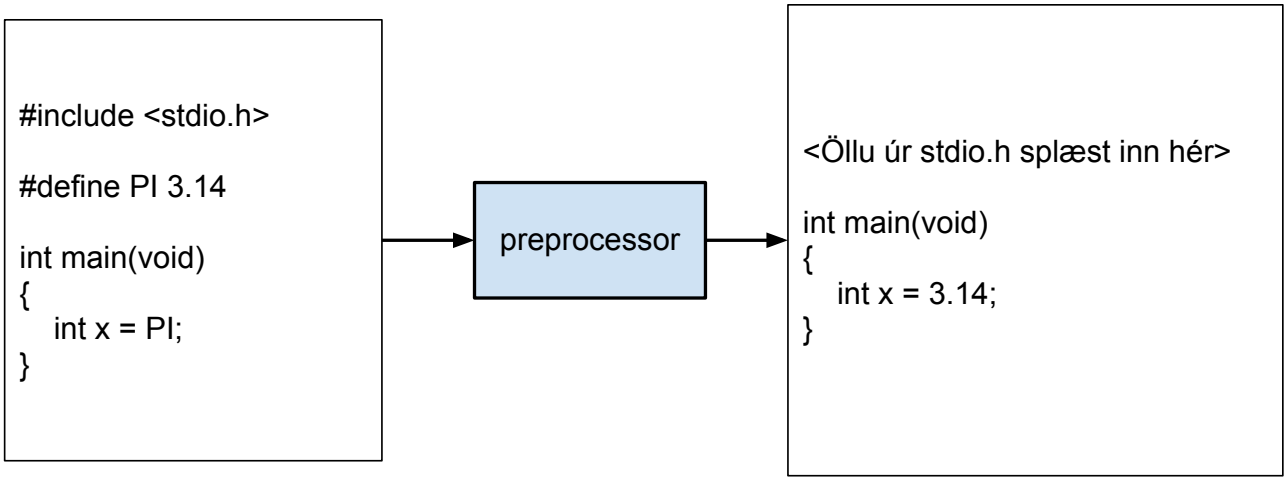
GCC



Athugasemdir

- Forþýðandi (preprocessor): Vinnur úr öllu sem byrjar á #
- Þýðandi (compiler): Þýðir sjálft forritið og býr til smalamálskóða
- Smali (assembler): Þýðir smalamál yfir á vélamál, býr til object skrár

Forþýðandi



Athugasemdir

- Forþýðandinn er heimskur, skiptir út `#define`, splæsir inn `#include`
- #include er notað til að deila skilgreiningum milli margra .c skráa
- #define er oft notað til að skilgreina fasta

Viðföng (argc, argv)

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;

    for (i = 0; i < argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);
}
```

Útskýringar

- Til að taka á móti viðföngum úr skipanalínu þá er main látið taka tvö viðföng: argc og argv
- argc er fjöldi viðfanga, argv er fylki strengja sem inniheldur viðföngin úr skipanalínu

Viðföng (argc, argv)

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;

    for (i = 0; i < argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);
}
```

Þýðing og keyrsla

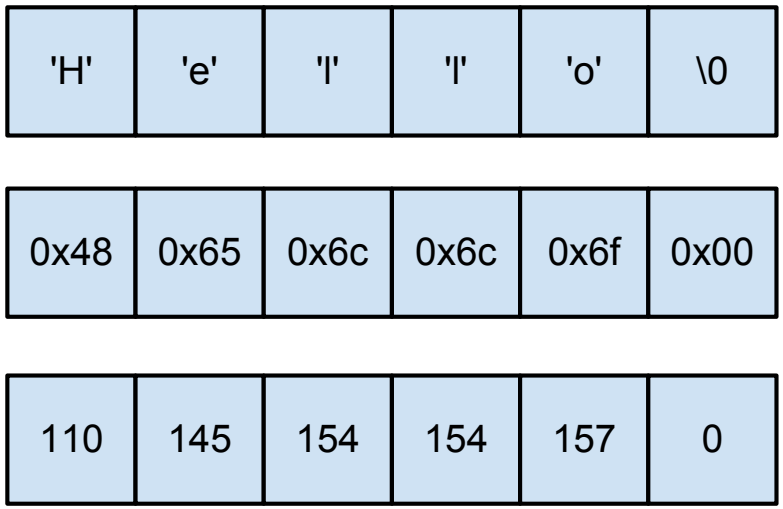
```
hhg@hhg:~/t2$ gcc argv.c -o argv
hhg@hhg:~/t2$ ./argv test1 test2
argv[0] = ./argv
argv[1] = test1
argv[2] = test2
hhg@hhg:~/t2$
```

Strengir í C

Strengir

- Það er ekkert sérstakt strengjatag í C, ólíkt Java
- Strengir eru táknaðir sem fylki af `char`
- Núll-bæti (`0x00`) táknar endalok strengs
- Tómi strengurinn `""` er fylki með eitt sæti sem inniheldur `0`

"Hello"



while í C

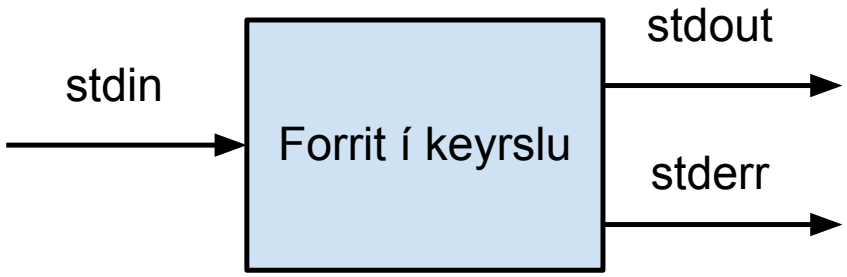
Kóði

```
while (C)
{
    S;
}
```

while í Java vs. C

- Java er með sérstakt boolean tag, lykkjuskilyrðið *C* verður að vera *boolean expression*, þ.e. segð með boolean útkomu
- C hefur ekki neitt sérstakt boolean tag, í stað þess er 0 tekið sem false og 1 sem true
- Endalaus lykkja í C er `while (1)`, en það er óleyfilegt forrit í Java því 1 er ekki boolean gildi

stdin, stdout, stderr



Straumar

- Þrír standard straumar: stdin (standard input), stdout (standard output), stderr (standard error)
- stdin vanalega tengt við lyklaborðið (eða lyklaborð á remote tölvu), en hægt að stýra því og tengja stdin við skrá á disk, socket út á netið, o.s.frv.
- Það sama gildir um alla aðra strauma

Lesa heiltölur af aðalinntaki

```c
#include <stdio.h>

int main(void)
{
    int n;

    while (1)
    {
        if (scanf("%d", &n) == EOF)
            break;

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

Lesa bókstafi af aðalinntaki

```c
#include <stdio.h>

int main(void)
{
    int c;

    while (1)
    {
        c = getchar();

        if (c == EOF)
            break;

        printf("%c\n", c);
    }
}
```

Prenta streng

```c
#include <stdio.h>
#include <string.h>

void print_chars(char *s)
{
    int n = strlen(s);
    int i;

    for (i = 0; i < n; i++)
        printf("s[%2d] = %c %3d 0x%02x\n", i, s[i], s[i], s[i]);
}

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        printf("usage: printchars <string>\n");
        return 1;
    }

    print_chars(argv[1]);
}
```

Dýnamískt minni

Almennt

- Engin sjálfvirk ruslasöfnun í C
- Handvirk ruslasöfnun gefur meiri stjórn og meira vald, en það kostar sitt
- Mikil vinna getur farið í að skilgreina ábyrgð á minni
- Tradeoff: tími tölvunnar vs. tími forritarans
- Í gamla daga voru tölvur mjög dýrar og tölvutími dýrari en tími forritarans, í dag hefur þetta snúist við

malloc og free

```
// allocates memory block of size n
void *malloc(size_t n);
// releases the memory block referenced by ptr
void free(void *ptr);
```