

# Introduktion till C – kort historia

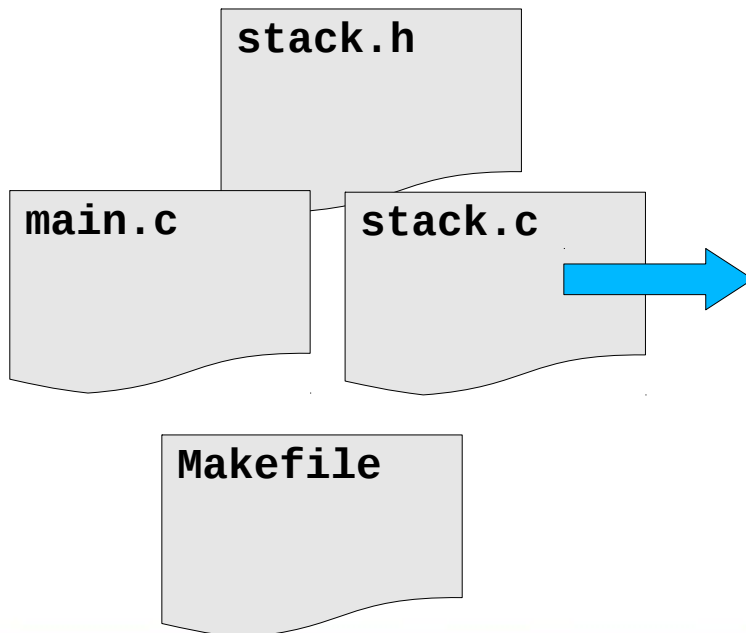
- Skapades i början av 1970-talet
- Ökade i popularitet i takt med operativsystemet Unix, vars “standardspråk” var/är C
- Ligger till grund för objektorienterade språk som C++ och Java
- “C is quirky, flawed, and an enormous success. While accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments.”
  - Dennis M. Ritchie (1941 - 2011),  
The development of the C language

# Varför C?

- Behärskar man C är det (relativt) lätt att lära sig andra “vanliga” programmeringsspråk
- Kompilatorer för så gott som alla datorarkitekturer  
=> Mycket *portabelt*
- Små och snabba program
- Lämpligt såväl för hårdvarunära som mera abstrakta uppgifter

# Källkod

- Program i C skrivs i textform – vilken texteditor som helst kan användas för detta ändamål
- Olika typer av filer:
  - **Källkod (\*.c), headerfiler (\*.h), makefiler**



```
stack.c — Kate
File Edit View Projects Bookmarks Sessions Tools Settings Help
New Open Save Save As Close Undo Redo
Documents
stack.c
1 #include "stack.h"
2
3 int stack_push(stack* s, int val)
4 {
5     if (s->top >= STACK_MAX_SIZE)
6     {
7         return -1;
8     }
9
10    s->array[s->top++] = val;
11    return 1;
12 }
```

Line 7, Column 18    INSERT    Soft Tabs: 4 (8)    UTF-8    C

Search and Replace    Terminal

# Kompilering av program

- En kompilator omvandlar källkodsinstruktioner till maskinkod för en viss datorarkitektur
  - T.ex. C-kod => maskinkod för x86- eller x64-processorer
- I Linuxmiljön använder vi oss av kompilatorn **gcc**
- Enklaste scenariot: Kompilatorn läser in **en** källkodsfil och skapar en körbar fil
- Mera realistisk: Ett antal egna filer läses in, kompileras, och *länkas* till yttre *programbibliotek*.
- Kompilatorn kan instrueras att *optimera* programmet med avseende på t.ex. storlek eller prestanda

# Programbibliotek

- Ineffektivt att uppfinna hjulet på nytt, dvs. implementera all funktionalitet som behövs “from scratch”
- Kan istället använda färdiga *programbibliotek* för t.ex. input/output, matematiska beräkningar, minneshantering...  
=> *Återanvändning* av kod (“Code reuse”)
- Biblioteken kan vara kommersiella eller fritt tillgängliga
- Under denna kurs kommer vi endast att använda oss av C:s standardbibliotek (ingår t.ex. i **gcc**)

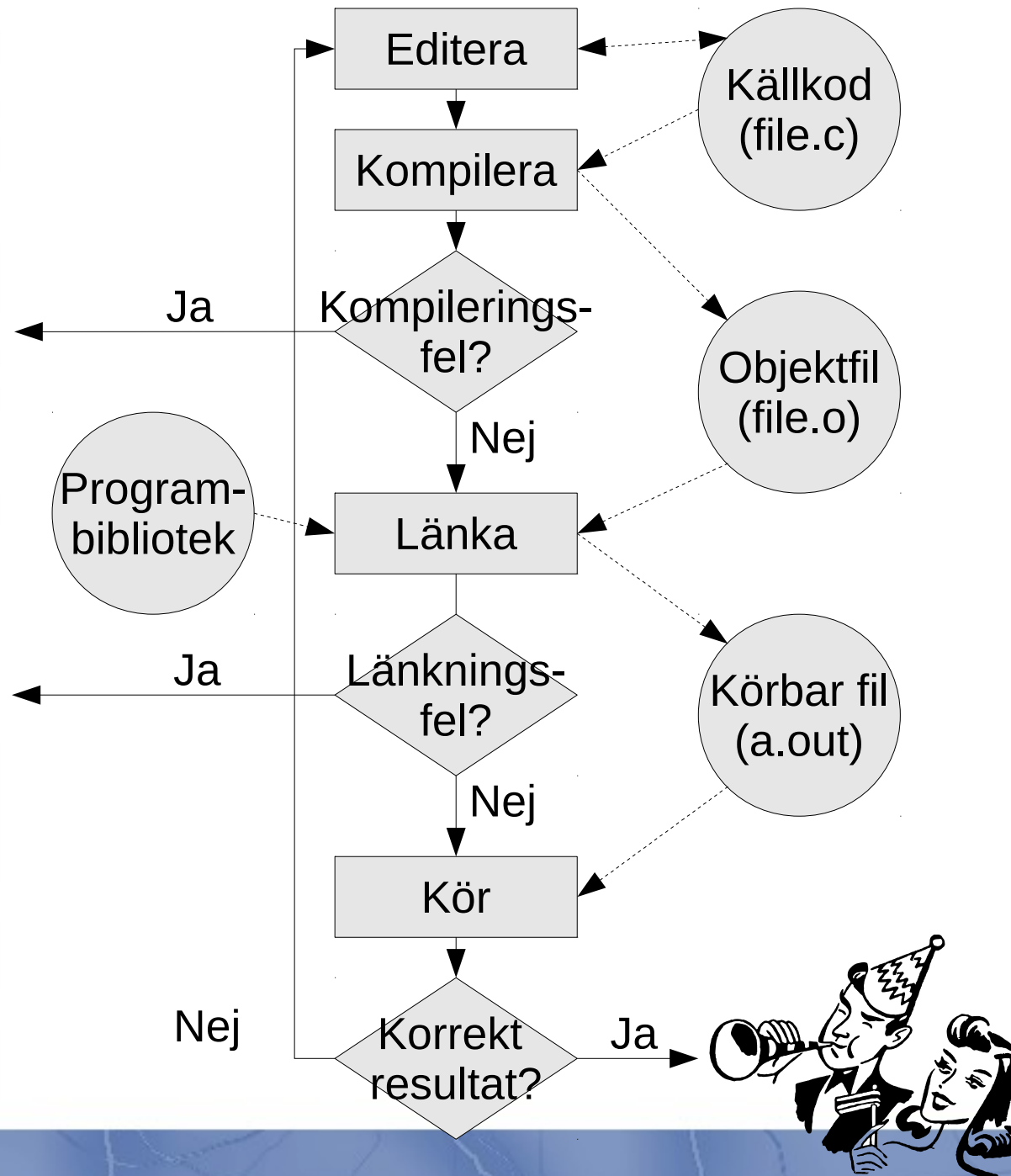
# “Programmeringsprocessen”

Exempel på kommandon i Linux-miljö:

*Editera:*  
**kate file.c**

*Kompilera & länka:*  
**gcc file.c**

*Köra programmet:*  
**./a.out**



# Vårt första program i C

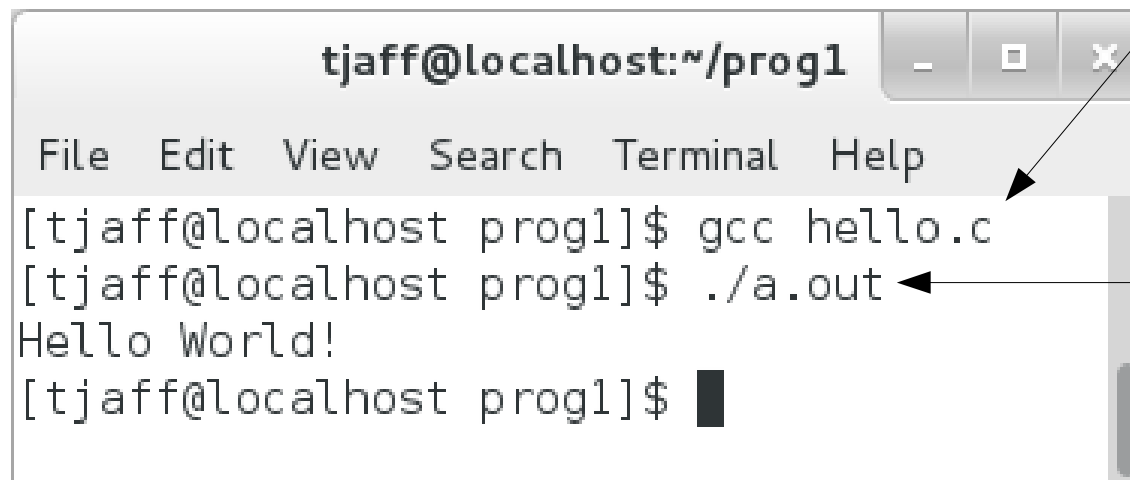
```
#include <stdio.h>

int main(void)
{
    printf("Hello World!\n");

    return 0;
}
```

Denna fil editeras i en text-editor och lagras under namnet **hello.c**

Programmet *kompileras*...



The screenshot shows a terminal window titled 'tjaff@localhost:~/prog1'. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The command history shows the user compiling 'hello.c' with 'gcc' and then running the resulting executable 'a.out'. The output of the program is 'Hello World!'.

```
tjaff@localhost:~/prog1
File Edit View Search Terminal Help
[tjaff@localhost prog1]$ gcc hello.c
[tjaff@localhost prog1]$ ./a.out
Hello World!
[tjaff@localhost prog1]$
```

...vilket resulterar i en *körbar fil* **a.out**



# Vårt första program i C

“Preprocessor directive” som instruerar kompilatorn att inkludera information från C:s standardbibliotek

```
#include <stdio.h>

int main(void)
{
    printf("Hello World!\n");

    return 0;
}
```

Detta är en funktion som heter **main**.

Den returnerar ett värde av typ **int** och accepterar inga parametrar (“**void** of parameters”)

**Viktigt:** Varje körbart program måste innehålla en funktion som heter **main** – detta är programmets startfunktion (“entry point”)



# Vårt första program i C

**printf** är en funktion i C:s standardbibliotek som skriver ut data till systemets standard output

```
#include <stdio.h>

int main(void)
{
    ▶ printf("Hello World!\n");

    return 0;
}
```

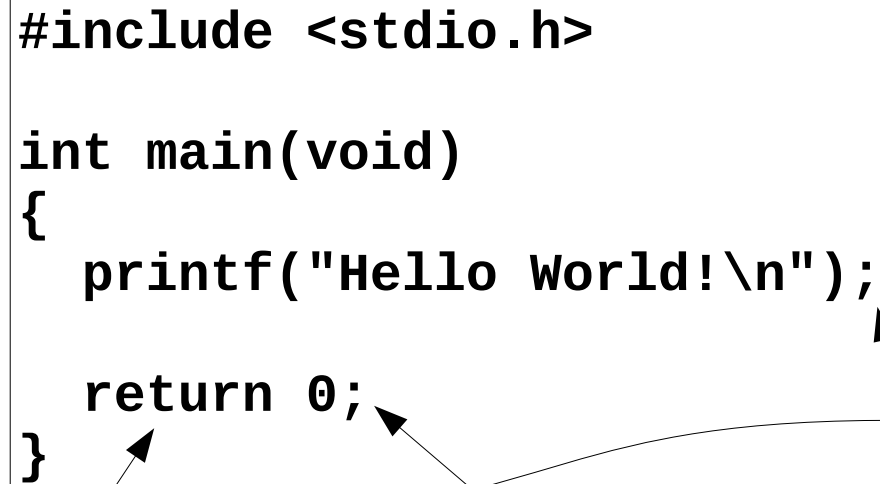
I motsats till **main** accepterar **printf** parametrar – i detta fall en teckensträng som inleds och avslutas med tecknet "

“Vågparenteserna” (*curly braces*) { och } visar var funktionen börjar och slutar. De kan också användas för att skapa *kodblock* inne i en funktion.

# Vårt första program i C

```
#include <stdio.h>

int main(void)
{
    printf("Hello World!\n");
    return 0;
}
```



**return** avslutar körningen av funktionen och returnerar ett värde (=0) till operativsystemet. En nolla innebär att programmet avslutades normalt.

Varje instruktion måste avslutas med ett semikolon ( ; )

# Praktiska råd och tips

- Planera programmet innan du börjar skriva kod
- Testkör ofta
- Gör säkerhetskopior
- Använd en enhetlig indentering
- Kommentera din kod
- Använd vettiga variabelnamn
- Gå igenom och korrigerar kompilatorns felmeddelanden en i taget, **uppfifrån och ner**
- Sätt in extra utskrifter för debuggning