Introduction to the C-Language and Programming Environment

Part 1: Basics to get started

C Language Tutorial
System Programming 251-0053
Winter Semester 2005/06

René Müller, IFW B45.2

Content

- Facts about C
- Writing and Compiling C Programs
- Tools
  - Compiler, Linker, Assembler, Debugger
- Working Environment

History of the C-Language

- C was developed by Dennis M. Ritchie at AT&T Bell Labs between 1969-1973
- C is a descendant of Ken Thompson's language "B"
- Ritchie and Brian Kernighan published the standard book "The C Programming Language" in 1978. The C-dialect book described in the original publication is now known as K&R-Style.
- Due to the ease of portability the language began to spread.
- However there was no precise language standard (K&R insufficiently precise)
- A slightly modified version of the language from K&R was standardised by ANSI 1989 (ANSI-C)
- The latest revision of the Standard is C99 (ISO)

Source: Ritchie D.M.: "The Development of the C Language", ACM SIGPLAN, 1993

The early days… the guys…

Dennis Ritchie (left) und Ken Thompson (right) in front
In front of a PDP-11 system with two text-terminals (1972)

source: Dennis Ritchie's home page, http://www.cs.bell-labs.com/who/dmr/

Their system… PDP-11

Digital Equipment Corporation (DEC) shipped first model of PDP-11 series in 1970.

CPU module of a PDP-11/20:
4kB memory, no floating-point unit, $12’000 (1972)

Additional core-memory module:
8 kB memory, $10‘000

source: http://www.psych.usyd.edu.au/pdp-11

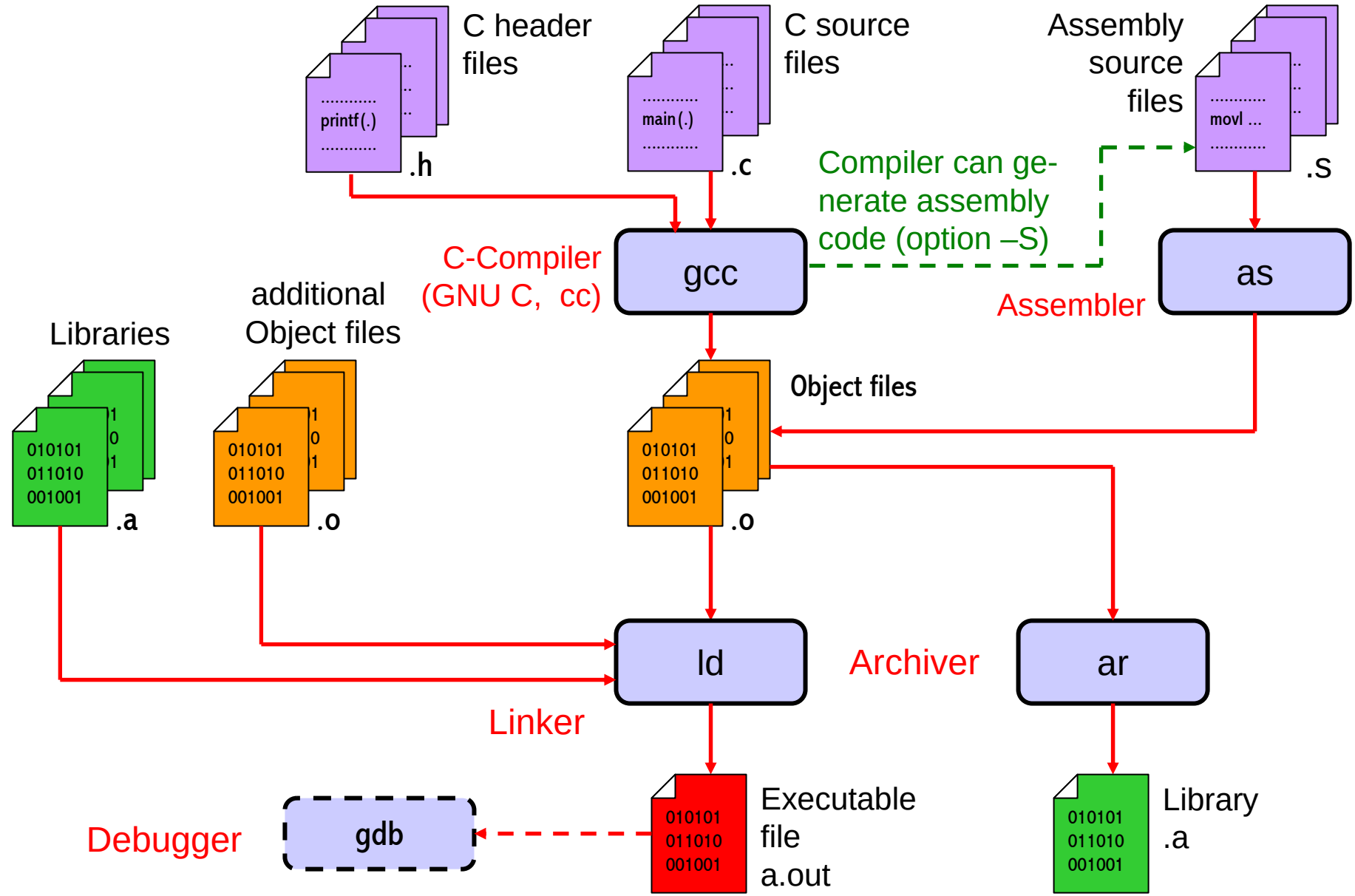
C the way should should not use it

This is a valid C program which happens to solve the n-Queens Problem.

```
v,i,j,k,l,s,a[99];
main()
{
   for(scanf("%d",&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<s&&(!k&&!!\
   printf(2+"\n\n%c"-(!l<<!j)," #Q"[l^v?(l^j)&1:2])&&++l||a[i]<s&&v\
   &&v-i+j&&v+i-j))&&!(l%=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i]);
}
```

Baruch Nissenbaum, Tel-Aviv University, Israel
Winner of the 7th Int. Obfuscated C Code Contest 1990
(www.ioccc.org)

From C code to an executable binary



Compiler Toolchain

- C-Compiler (**gcc** von GNU, or cc von DEC, SUN, etc.) reads C source file and
  - Creates assembly file (.s) (gcc –S)
  - But can also
    - create object file (.o) by calling Assembler (gcc –c)
    - create executable file by calling the Linker (default)

  afterwards
- Assembler (**as**) generates object files (.o) that contain machine code from assembly source files (.s)
- Linker (**ld**) collects object files and libraries and creates executable file
- Archiver (**ar**) creates libraries (.a) from object files
- Debugger (**gdb** from GNU) allows inspection of memory and register content of a running program

„Hello World“-Example in C

Save the code in a file ‘hello.c‘:

```
main()
{
    printf(“Hello World!\n“);
}
```

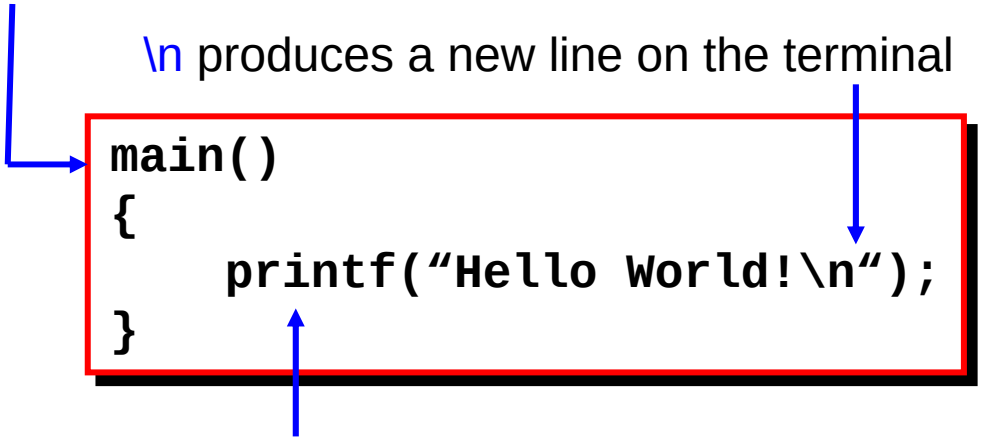
From the terminal the executable can be built and started:

```
$ ls
hello.c
$ gcc -o hello hello.c
$ ls
hello.c  hello
$ ./hello
Hello World!
$
```

„Hello World“-Example in C (2)

The execution of the program starts by calling the main-function.

\n produces a new line on the terminal



Writes string to standard output of the terminal.
printf is a C-Function, which is part of the standard C-library.

The main function can have different signatures:

- **main()**
- **int main(int argc, char **argv)**
- **int main(int argc, char **argv, char **env)**

Compiling and Linking

`gcc -o hello hello.c` directly creates an executable file **`hello`** (specified by option –o) from the source file. Note: without specifying option -o, gcc chooses the default name a.out for the output file. The creation of the machine code and linking is done automatically.
Calling gcc with option –v (verbose) reveals what gcc does:

```
$ gcc –v –o hello hello.c
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/specs
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared --enable-
threads=posix --disable-checking --with-system-zlib --enable-__cxa_atexit --host=i386-redhat-linux
Thread model: posix
gcc version 3.2.3 20030502 (Red Hat Linux 3.2.3-53)
 /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/cc1 -lang-c -v -D__GNUC__=3 -D__GNUC_MINOR__=2 -D__GNUC_PATCHLEVEL__=3 -
D__GXX_ABI_VERSION=102 -D__ELF__ -Dunix -D__gnu_linux__ -Dlinux -D__ELF__ -D__unix__ -D__gnu_linux__ -D__linux__ -D__unix -
D__linux -Asystem=posix -D__NO_INLINE__ -D__STDC_HOSTED__=1 -Acpu=i386 -Amachine=i386 -Di386 -D__i386 -D__i386__ -
D__tune_i386__ hello.c -quiet -dumpbase hello.c -version -o /tmp/ccln1xyb.s
GNU CPP version 3.2.3 20030502 (Red Hat Linux 3.2.3-53) (cpplib) (i386 Linux/ELF)
GNU C version 3.2.3 20030502 (Red Hat Linux 3.2.3-53) (i386-redhat-linux)
        compiled by GNU C version 3.2.3 20030502 (Red Hat Linux 3.2.3-53).
#include "..." search starts here:
#include <...> search starts here:
 /usr/local/include
 /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/include
 /usr/include
End of search list.
 as -V -Qy -o /tmp/ccGJgvrg.o /tmp/ccln1xyb.s
GNU assembler version 2.14.90.0.4 (i386-redhat-linux) using BFD version 2.14.90.0.4 20030523
 /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/collect2 --eh-frame-hdr -m elf_i386 -dynamic-linker /lib/ld-linux.so.2 -o hello
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/../../../crt1.o /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/../../../crti.o
/usr/lib/gcc-lib/i386-redhat-linux/3.2.3/crtbegin.o -L/usr/lib/gcc-lib/i386-redhat-linux/3.2.3 -L/usr/lib/gcc-lib/i386-
redhat-linux/3.2.3/../../.. /tmp/ccGJgvrg.o -lgcc -lgcc_eh -lc -lgcc -lgcc_eh /usr/lib/gcc-lib/i386-redhat-
linux/3.2.3/crtend.o /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/../../../crtn.o
```

Debugging with GNU-Debugger

When command line option -g is specified, gcc puts additional information (symbols, name and addresses of variables and functions) into the compiled file. These symbols can be used by a debugger to show meaningful names instead of numbers.

Example with gdb (GNU debugger):

```
$ gcc -g -o hello hello.c
```

```
1: /* hello.c */
2: main()
3: {
4:   int i=0;
5:   while (i<3) {
6:      printf("Hello World!\n");
7:      i = i+1;
8:   }
9: }
```

Debugging with GNU-Debugger (2)

```
$ gdb hello
(gdb) list
1       main()
2       {
3          int i;
4          while (i<3) {
5             printf("Hello World!\n");
6             i = i+1;
7          }
8       }
(gdb) break 5
Breakpoint 1 at 0x804835c: file hello.c, line 5.
(gdb) run
Breakpoint 1, main () at hello.c:5
5   printf("Hello World!\n");
(gdb) print i
$1 = 0
(gdb) cont
Continuing.
Hello World!
Breakpoint 1, main () at hello.c:5
5     printf("Hello World!\n");
(gdb) print i
$2 = 1
(gdb) quit
```

Working Environment – UNIX Command Line

- Since we are using Linux on command line (terminal) you have the following options:
- ETH Machines
  - locally at ETH (IFW C31, IFW D31, IFW D35)
    make sure you choose linux when starting the system, login with your n.ethz account
  - remote access to machines at ETH machines rifpc{1-30}.ethz.ch
    - from Linux over ssh
      $ ssh yourlogin@rifpc{1-30}.ethz.ch
    - from a Windows machine
      Get PuTTY a simple SSH client
      http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
    - Note: there might be some firewall issues when accessing from outside ETH, in those cases use VPN.
- Personal notebook
  - install Linux – generally considered a good idea ;-)
  - install GCC-environment from Cygwin under Windows
    http://www.cygwin.com

Command Line Basics

- Change directory '**cd <dirname>**'
  - '**cd ..**' change to parent directory
  - '**cd ~**' change to home directory
  - '**cd foo/bar**' change to directory bar (which is subdirectory of foo) from current directory
- List directory content '**ls**' for short listing and '**ls –la**' for long listing
- Create subdirectory '**mkdir <dirname>**'
- Delete files '**rm file**'
  - Delete files and directories recursively: '**rm –r <dirname>**'
- Delete empty directory '**rmdir dirname**'
- Concatenate 3 files to outfile '**cat file1 file2 file > outfile**'
- Tab-completion (<TAB> key auto completes command line)

  $ ls
  bar foo
  $ cd b<TAB> → by auto completion → **cd bar**
  - Very helpful, makes working on command line even more efficient

Man Pages

- Man pages are the online help for various UNIX tools
  $ man gcc → Shows man page for GCC
  - Use <space> and/or <up>/<down> keys to move
  - Key <q> returns back to command line
  - Use man if you know the name of the tool you want to get the man page
- Apropos searches description

```
$ apropos compiler
g77   (1)   GNU project Fortran 77 compiler
gcc   (1)   GNU project C and C++ compiler
gcj   (1)   Ahead-of-time compiler for Java
less  (8pm) perl pragma to request less of something
```

  $ man less → shows "less – opposite of more" (section 1)
  $ man 1 less → shows also "less – opposite of more" (section 1)
  $ man 8pm less → shows "perl pragma to …" (section 8pm)

- If '**man gcc**' yields an error "No manual entry for gcc" make sure that the environment variable is properly set.

```
$ setenv MANPATH /usr/sepp/man:/usr/man:/usr/share/man:$MANPATH
```