Programming in C autumn 2010

Päivi Kuuppelomäki

Week 1

The structure of the course

- Lectures: thu 9-10
- Excercises
- Study on your own
- Project
- Course exam
- Course book:

  Müldner: C for java programmers

Week scedule

- Week 1 – compiling, linking, types, structures, macros
- Week 2 – text files, funktions
- Week 3 – pointers
- Week 4 – structures and unions
- Week 5 – strings, arrays
- Week 6 – modules and libraries

Exercises and studing on your own

- Exercises are available on course page
- Lectures cover difficult things from the course and others should be studied on your own

Project work

- List of Project work will be available later from the course page
- Work should be returned at latest at the end of the first week during period II.
- You should returned one module from your project earlier and you get feedback from other students

Course exam

- Thu 21.10. 16-19 room A111 (CHECK!)
- What kind of tasks
  - Sama kind of tasks as in exercises
  - Do a program
  - ”What errors are in aprogram”
  - etc.
- Important to know: pointers, files, arrays, structures, strings, command line parameters

Principles of C-language

Programmer knows what she/or he does!

- Language does not prevent ”bad things” – Programmer might write a cryptic code
- Erros that came by using careless programming might take time to find out
- No object, that hides structures
- Pointers important part of a language
- Is suitable near machine level programming, because it is possible to compile C-programs to efficient code
- For example linux has been coded using C

Comparison of C and Java

- *primitive data types*: character, integer, and real
  - In C, they are of different sizes,
  - there is no Unicode 16-bit character set
- *structured data types*: arrays, structures and unions.
  - In C, arrays are static
  - there are no classes
- *Control structures* are similar
- *Functions* are similar

Comparison of C and Java

- Java references are called pointers in C.
- Java constructs missing in C:
  - packages
  - threads
  - exception handling
  - garbage collection
  - standard Graphical User Interface (GUI)
  - built-in definition of a string
  - standard support for networking
  - support for program safety.

Programming style

- Try to write clear code and use style your have learned during Java courses
- Your do not get extra points by writing short and cryptic code

```
do {
  if (scanf(”%d”, &i) !=1 ||
       i == SENTINEL)
    break;
  if (i>maxi)
    maxi = i;
} while (1);
```

```
void show (char *p) {
  char *q;
  printf(”[ ”);
  for (q=p; *q != ’\0’; q++)
    printf(”%c ”, *q);
  printf(”]\n”);
}
```

Programming process

- Write a progam
  - Use editor
- Compile it
  - Choose a right compiler
- Linking
  - Compiled programming module is linked to other modules
- Run it
  - Run the program

Writing a program

- Program should generate an ordinary text file ,

```
int main (void)
{
  printf(”Hello world \n”);
  return 0;
}
```

- Possible programs
  - emacs: uses own window
    - Remember to run from the command line using emacs & so you do not preserve command interpreter
  - Kate
- Learn by yourself

Compiling

- Department's Linux environment has gcc (also cc works)

```
kuuppelo@wrl-130:~$ which gcc
/usr/bin/gcc
kuuppelo@wrl-130:~$ ls -l /usr/bin/gcc
-rwxr-xr-x 2 root root 195844 May 26 02:34 /usr/bin/gcc*
kuuppelo@wrl-130:~$ gcc -dumpversion
4.1.2
```

gcc --help

```
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  - -help                  Display this information
  - -target-help           Display target specific command line options
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search path
  -print-libgcc-file-name  Display the name of the compiler's companion library
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component <prog>
  -print-multi-directory   Display the root directory for versions of libgcc
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -Wa,<options>            Pass comma-separated <options> on to the assembler
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor
  -Wl,<options>            Pass comma-separated <options> on to the linker
  -Xassembler <arg>        Pass <arg> on to the assembler
  -Xpreprocessor <arg>     Pass <arg> on to the preprocessor
  -Xlinker <arg>           Pass <arg> on to the linker
```

gcc –help (continues)

```
  -save-temps             Do not delete intermediate files
  -pipe                   Use pipes rather than intermediate files
  -time                   Time the execution of each subprocess
  -specs=<file>           Override built-in specs with the contents of <file>
  -std=<standard>         Assume that the input sources are for <standard>
  -B <directory>          Add <directory> to the compiler's search paths
  -b <machine>            Run gcc for target <machine>, if installed
  -V <version>            Run gcc version number <version>, if installed
  -v                      Display the programs invoked by the compiler
  -###                    Like -v but options quoted and commands not executed
  -E                      Preprocess only; do not compile, assemble or link
  -S                      Compile only; do not assemble or link
  -c                      Compile and assemble, but do not link
  -o <file>               Place the output into <file>
  -x <language>           Specify the language of the following input files
                          Permissible languages include: c c++ assembler none
                          'none' means revert to the default behavior of
                          guessing the language based on the file's extension

Options starting with -g, -f, -m, -O, -W, or --param are automatically
 passed on to the various sub-processes invoked by gcc.  In order to pass
 other options on to these processes the -W<letter> options must be used.
```

Compiling

- Compiling

  gcc helloworld.c

  or

  gcc –o helloworld |

  helloworld.c

- Tässä tehdään
  - preprosseing
  - compiling and
  - linking

```
int main (void)
{
  printf(”Hello world \n”);
  return 0;
}
```

- Result is a runnable file

  a.out

  or

  helloworld

gcc -v helloworld.c

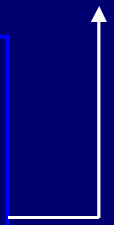
```
Reading specs from /usr/lib/gcc/i386-redhat-linux/3.4.2/specs
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --
      enable-shared --enable-threads=posix--disable-checking --with-system-zlib --enable-__cxa_atexit
      --disable-libunwind-exceptions --enable-java-awt=gtk --host=i386-redhat-linux
Thread model: posix
gcc version 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)
 /usr/libexec/gcc/i386-redhat-linux/3.4.2/cc1 -quiet -v helloworld.c -quiet -dumpbase helloworld.c -
      auxbase helloworld -version -o /tmp/niklande/cc1k6oOu.s
ignoring nonexistent directory "/usr/lib/gcc/i386-redhat-linux/3.4.2/../../../../i386-redhat-linux/include"
#include "..." search starts here:
#include <...> search starts here:
→ /usr/local/include
 /usr/lib/gcc/i386-redhat-linux/3.4.2/include
→ /usr/include
End of search list.
GNU C version 3.4.2 20041017 (Red Hat 3.4.2-6.fc3) (i386-redhat-linux)
        compiled by GNU C version 3.4.2 20041017 (Red Hat 3.4.2-6.fc3).
GGC heuristics: --param ggc-min-expand=98 --param ggc-min-heapsize=129136
 as -V -Qy -o /tmp/niklande/ccQshiJR.o /tmp/niklande/cc1k6oOu.s
GNU assembler version 2.15.90.0.3 (i386-redhat-linux) using BFD version 2.15.90.0.3 20040415
 /usr/libexec/gcc/i386-redhat-linux/3.4.2/collect2 --eh-frame-hdr -m elf_i386 -dynamic-linker /lib/ld-
      linux.so.2 /usr/lib/gcc/i386-redhat-linux/3.4.2/../../../crt1.o /usr/lib/gcc/i386-redhat-
      linux/3.4.2/../../../crti.o /usr/lib/gcc/i386-redhat-linux/3.4.2/crtbegin.o -L/usr/lib/gcc/i386-redhat-
      linux/3.4.2 -L/usr/lib/gcc/i386-redhat-linux/3.4.2 -L/usr/lib/gcc/i386-redhat-linux/3.4.2/../../..
      /tmp/niklande/ccQshiJR.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed -lgcc_s -
      -no-as-needed /usr/lib/gcc/i386-redhat-linux/3.4.2/crtend.o /usr/lib/gcc/i386-redhat-
      linux/3.4.2/../../../crtn.o
```

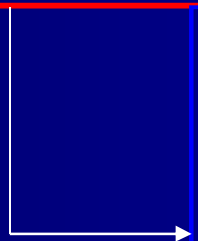
gcc -ansi -pedantic -Wall

- By using optios –Wall and –pedantic a compiler gives more warnings
- Option –ansi assures that a compiler will use ansi standard

```
gcc -ansi -pedantic -Wall -o helloworld helloworld.c
helloworld.c: In function 'main':
helloworld.c:3: warning: implicit declaration of function 'printf'
helloworld.c:3: warning: incompatible implicit declaration of built-in function'prin
```

```
int main (void)
{
  printf(”Hello world \n”);
  return 0;
}
```





```
#include <stdio.h>
int main (void)
{
  printf(”Hello world \n”);
  return 0;
}
```

Program having several modules

- Each module, compiling unit, library in its own file
- Compiling separately

  gcc –c main.c
- Linking together

  gcc –o main.o eka.o toka.o

Program having several modules

```
/* main.c */
#include <stdio.h>
#include ”eka.h”
#include ”toka.h”
int main (void)
{
  eka(); toka ();
  return 0;
}
```

```
/* eka.c */
#include <stdio.h>
#include ”eka.h”
void eka (void)
{
  puts(” eka ”);
}
```

```
/* toka.c */
#include <stdio.h>
#include ”toka.h”
void toka (void)
{
  puts(” toka ”);
}
```

```
/* eka.h */

void eka (void);
```

```
/* toka.h */

void toka (void);
```

```
gcc –c main.c
gcc –c eka.c
gcc –c toka.c
gcc –o ohjelma main.o eka.o toka.o
```

Compiling modules – make

- It is not practical to type long commands
- Use file Makefile
- Runnable commands should be written as rules into a file

```
target: files needed
    command1
    command2
    ..
    commandy
```

- Please note that commands are indented by using tab not spaces!

makefile

```
gcc –c main.c
gcc –c eka.c
gcc –c toka.c
gcc –o ohjelma main.o eka.o toka.o
```

make

- Write a file makefile once
- Use it several times by giving command make

```
# makefile
CC = gcc –ansi –pedantic –Wall
ohjelma: main.o eka.o toka.o
        $(CC) –o ohjelma main.o eka.o toka.o
eka.o: eka.c eka.h
        $(CC) –c eka.c
toka.o: toka.c toka.h
        $(CC) –c toka.c
main.o: main.c eka.h toka.h
        $(CC) –c main.c
```

make --help

```
Usage: make [options] [target] ...
Options:
  -b, -m                      Ignored for compatibility.
  -C DIRECTORY, --directory=DIRECTORY
                              Change to DIRECTORY before doing anything.
  -d                          Print lots of debugging information.
  --debug[=FLAGS]             Print various types of debugging information.
  -e, --environment-overrides
                              Environment variables override makefiles.
  -f FILE, --file=FILE, --makefile=FILE
                              Read FILE as a makefile.
  -h, --help                  Print this message and exit.
  -i, --ignore-errors         Ignore errors from commands.
  -I DIRECTORY, --include-dir=DIRECTORY
                              Search DIRECTORY for included makefiles.
  -j [N], --jobs[=N]          Allow N jobs at once; infinite jobs with no arg.
  -k, --keep-going            Keep going when some targets can't be made.
  -l [N], --load-average[=N], --max-load[=N]
                              Don't start multiple jobs unless load is below N.
```

make --help (continues)

```
-n, --just-print, --dry-run, --recon  Don't actually run any commands; just print
   them.
-o FILE, --old-file=FILE, --assume-old=FILE
                        Consider FILE to be very old and don't remake it.
-p, --print-data-base       Print make's internal database.
-q, --question              Run no commands; exit status says if up to date.
-r, --no-builtin-rules      Disable the built-in implicit rules.
-R, --no-builtin-variables  Disable the built-in variable settings.
-s, --silent, --quiet       Don't echo commands.
-S, --no-keep-going, --stop
                        Turns off -k.
-t, --touch                 Touch targets instead of remaking them.
-v, --version               Print the version number of make and exit.
-w, --print-directory       Print the current directory.
--no-print-directory        Turn off -w, even if it was turned on implicitly.
-W FILE, --what-if=FILE, --new-file=FILE, --assume-new=FILE
                        Consider FILE to be infinitely new.
--warn-undefined-variables  Warn when an undefined variable is referenced.
```

After compiling (and linking)

- We have a runnable program, but does it work?
- Try and test
- Search errors
  - Print soming that helps you to understand program
  - Write code and think
  - Use debugger
- Analyse how well the test cover different situations (Other courses teach how)

Testing

- Try to find errors
- Use different kind of inputs
- You can automate tests (for example using skripts etc.)

  This is out of scope of this course
- During this course it is enough
  - Right and wrong values of inputs
  - Typical values near limits (-1,0,1)

Print to help

- printf (”Fname: Name of a varible %d \n”, variable);
- Try to find out how the program is working in an error situation
- Add some print statements near error point
- Often easier to use than the debugger, if there is a clue where the error is

Debugger gdb

(gdb) help

List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

- Compiling using option -g

core dump

- Crashing program creates often a core dump where is the state of the memeory and registers during the time program crashed
- You can look att the core dump uding debugger and it might be possible to look at the values of varibles and/or find out were program was when it crashed