Programming in C
autumn 2010

Päivi Kuuppelomäki
Week 1
The structure of the course

- Lectures: thu 9-10
- Exercises
- Study on your own
- Project
- Course exam
- Course book:
  Müldner: C for java programmers
Week schedule

- 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 studying on your own

- Exercises are available on course page
- Lectures cover difficult things from the course and others should be studied on your own
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 return 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 a program"
  - 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
  - Errors 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 you have learned during Java courses.
- Your do not get extra points by writing short and cryptic code.

```c
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 program
  - 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

- 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

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

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

```bash
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
```
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
- -WI,<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)

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

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
  - preprossing
  - compiling and
  - linking

- Result is a runnable file
  - `a.out`
  - `helloworld`

```c
int main (void)
{
  printf("Hello world \n");
  return 0;
}
```
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-linux/3.4.2/cc1 -quiet -dumpbase helloworld.c -auxbase helloworld -version -o /tmp/niklande/cc1k6oOu.so

ignoring nonexistent directory "/usr/lib/gcc/i386-redhat-linux/3.4.2/../../../../i386-linux/include"
#include "..." search starts here:
#include <...> search starts here:
/usr/local/include
/usr/lib/gcc/i386-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)
GNU assembler version 2.15.90.0.3 (i386-redhat-linux) using BFD version 2.15.90.0.3 20040415
/usr/libexec/gcc/i386-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/libgcc/i386-redhat-linux/3.4.2/crtbegin.o -L/usr/lib/gcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/gcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2 -L/usr/lib/libgcc/i386-redhat-linux/3.4.2
gcc -ansi -pedantic -Wall

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

```
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 ");
}

/* eka.h */
void eka (void);

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

/* 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!
Write a file `makefile` once

Use it several times by giving command `make`

```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
gcc –c main.c
gcc –c eka.c
gcc –c toka.c
gcc –o ohjelma main.o eka.o toka.o
```
Usage: make [options] [target] ...

Options:
- `--help` Print this message and exit.
- `--ignore-errors` Ignore errors from commands.
- `--jobs[=N]` Allow N jobs at once; infinite jobs with no arg.
- `--keep-going` Keep going when some targets can't be made.
- `--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 something 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 variable %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
A core dump is created when a program crashes. It contains the state of the memory and registers during the time the program crashed. By analyzing the core dump, it might be possible to determine the values of variables and/or ascertain where the program was when it crashed.