CS Basics - Exercises
Basics of the C language

E. Benoist and P. Mainini

Fall Term 2022-23

1 Write a First C Program

Based on the examples given, write a program called `circle`, which computes the *area and perimeter given the radius of a circle.* The radius is entered as a number on standard input (you can read it using `scanf()`).

You should define a symbolic constant `PI` and make use of the relevant libraries, e.g. `stdio.h` and `math.h`.

Also write a corresponding `Makefile` for your program. Include build targets for development (e.g. no optimization and with debug symbols) and for release (e.g. with optimization, no debug symbols).

2 Debug Your Program Using `gdb`

Debug your program from the previous exercise using `gdb` ("`gdb circle`").

Note: It is recommended to switch debugging settings from assembly to C, e.g. using "`layout bti1021c`". You may now also change the default in your `$HOME/.gdbinit`.

You may for example use the following commands in `gdb`:

- `run` to start the program
- `break xx` where xx is the number of the line where you want to set a break point. You can also set a break point on a function name (e.g. `main`)
- `print varname` (or `p varname`) to see the value of the variable called `varname`
- `list` to see where you are inside the code
- `step` (or `s`) to go one instruction forward
- `next` (or `n`) the same, without following into called functions
- `continue` to go to the next break point

- `clear` to remove a breakpoints
- `quit` to exit the debugger.

Be sure to debug your program using both builds, with and without debug information (see previous exercise). What is the difference?

Clearly, `ddd` *and other front-ends for* `gdb` *may still be used. You can then also use* `print` *inside the gdb console, to see the value of a variable.*

Refer to the provided `gdb-cheat-sheet.md` *for more details about* `gdb` *commands.*

3 Use the GNU Build System for Building Your Program

Adjust your program to use the GNU Build System ("autotools") for building (and distribution etc.. . .):

1. Create `Makefile.am` (see below)
2. Run "`autoscan`" to generate `configure.scan`
3. Copy or rename `configure.scan` to `configure.ac`
4. Adjust `configure.ac`
   - Adjust the line "`AC_INIT`" with name, version and author of your program
   - Add a line "`AM_INIT_AUTOMAKE`" in section "`# Checks for programs.`"
5. Create additional, required files: `AUTHORS`, `ChangeLog`, `NEWS` and `README`
6. Generate `configure` script using "`autoreconf -fi`"

You may now generate a `Makefile` for your program by running "`./configure`". For this exercise, we recommend to run "`./configure --prefix=$HOME`", which will prepare the `Makefile` to install the program in your home directory.

Finally, try building your program using "`make`" and install it using "`make install`". Explore other options of automake, e.g. "`make uninstall`", "`make dist-gzip`". . .

3.1 Example for `Makefile.am`

```
bin_PROGRAMS = \
    circle
circle_SOURCES = \
    circle.c
```

4 Read a Bit of Documentation

In order to learn more about the tools used in this part of the course, we recommend to get an overview of the relevant documentation:

- The GCC manual: `https://gcc.gnu.org/onlinedocs/`
- GNU make: `https://www.gnu.org/software/make/manual/html_node/index.html`
- GNU automake: `https://www.gnu.org/software/automake/manual/automake.html`