How to survive ARQCP

2024/2025

Notes:

- This document is an incomplete short set of tips, that should be completed with lots of practice, by solving the exercises.

1 Linux Usage

- You can use more than one window at the same time without closing them (ex: one terminal window for your programs and another for running the tests). You can/should use one window for the editor and another for compiling.
- Choose an editor with syntax highlighting/help and with line numbers.
- Use an editor cheat sheet to know the editor commands.
- On antiX the following editores are installed: `emacs`, `geany`, `joe`, `micro`, `nano`, `pico`, e `vim` (in alphabetical order).
- To interrupt the execution of a program use `Ctrl-C`. If you use `Ctrl-Z` you will only suspend the execution of the program. To resume the execution of a program you can use the command `fg`.
- You can change between different programs on the same terminal with `tmux`[1], and you can have more than one tab on one terminal.

2 Compilation

- Write always a `Makefile` (file) to compile using `make`.
- Turn on always a good set of warnings (`-Wall -Wextra -fanalyzer -g`).
- Watch carefully the commands that run when compiling a program.
- Start correcting your programs always by the the first error (not the last).
- Look at the source code and at the error messages at the same time.
- Some editors/plugins do syntax checking when saving a file.
- C/C++ source code can be checked using the command: `cppcheck *.c`

[1]`https://github.com/tmux/tmux/wiki`

3 Running a program

- Before running a program correct all the errors and all the warnings.
- As you will be running many times your programs, if they don't have any data input, they will be easier/faster to run.
- The unit tests test not only the return value of the functions, but also their behaviour.
- When a test fails, it prints the source code line where it fails.
- If a program passes the tests, this does not means the program is correct. It only means that the program passes that set of tests.
- You can see memory errors (and where a program "crashes") using valgrind: `valgrind ./prog`
- You can debug a program using gdb (or in graphical mode with ddd): `gdb -tui prog`
- The gdb debugger does not work correctly if the programs are not compiled with the `-g` flag, or when assembly code sections are not correctly defined.

4 Common errors

- Uninitialised variables in C.
- Be careful when using `sizeof`, as it is an operator that works at compile time, and when applied to a pointer returns the size of the pointer.
- Incorrect usage of strings (attribution, comparison):

  ```
  str="One";   // wrong!
  ...
  if (str == "") // wrong!
  ...
  ```

- You can however initialise a string, and there should be space for the final 0:

  ```
  char str2[]="Two";  /* the same as: char str2[4]="Two"; */
  ```

- This is the initialization of a pointer to a constant string, that can be read (but cannot be written into):

  ```
  char* str3="Three";
  ```

- While copying a string the final 0 should always be copied.
- The `*` is not part of the pointer name, as it is an operator. A common mistake is:

  ```
  *ptr++;   // wrong!
  ```

- This C statement does not do anything:

  ```
  str - string_size; // wrong!
  ```

- When programming in Assembly respect the correct register usage conventions (as tests will crash).
- Be aware of the microprocessor programming model. Example: the `%al` register is part of the `%ax` register, that is part of the `%eax` register, that is part of the `%rax` register.
- Please dont mix 'l' with '1': don't write `mov1` instead of `movl`