Informatik und Angewandte Kognitionswissenschaft
Lehrstuhl für Hochleistungsrechnen

Thomas Fogal
Prof. Dr. Jens Krüger

High-Performance Computing

http://hpc.uni-due.de/teaching/wt2014/nbody.html

Exercise 0 (10 Points)

All assignments should be pushed to your personal Git repository. Assignments are due at midnight on the due date. No late assignments will be accepted.

All assignments must include a `Makefile` for compiling your assignments. Assignments which do not compile will receive 0 points. Assignments that do not satisfy the test inputs will receive 0 points.

Please **do not include output** other than what was requested by the assignment details.

Your assignment will be graded on the `duecray.uni-due.de` supercomputer. It does not matter if your program runs correctly on another machine; it must run correctly on `duecray` to receive credit.

This assignment has many purposes: requesting a user account for future use, learning the basics of C programming, and learning this course's unique submission and grading system.

1 Requesting an account (0 points)

We will use a local supercomputer for this course. You will need to access the machine remotely, via ssh.

Not all students have access to this machine: your account must be enabled for it. I will request an account on your behalf, but I require your student information.

Send me (thomas.fogal at uni-due) an email **from your uni-due account** with your name, uni-due username, and matriculation number **by Friday, October 24th**.

Note that you may have already done this in class during the kick off meeting.

2 Averaging (10 points)

Your task is to write a C99 program that computes the average of the integer values given on standard input. The number of values to be input is variable. Your program should continue reading values until it receives `EOF` during a read. The program should then output a single line, the floating point average of all values given.

You may assume that there is at least one value given as input.

A sample session is given in Listing 1:

```
$ ./avg
1
2
1.500000
$ echo "1 2" | ./avg
1.500000
$ echo "1 2 3 4 5" | ./avg
3.000000
$ echo "30 45" | ./avg
37.500000
$ echo "36 42 943  323 23 23  45 903 245" | ./avg
287.000000
```

Listing 1: Sample runs of `avg` program.

Your submission must include a `makefile` that builds your code. The makefile should build an executable named `avg`. We will use GNU make in this class. You can learn the basics of make by asking Tom for an introduction or through simple web searches. You might want to start with the `makefile` given in Listing 2.

```
CFLAGS:=-std=c99 -Wall -Wextra
all: simple.o avg

avg: simple.o
  $(CC) $(CFLAGS) $^ -o $@

clean:
  rm -f avg simple.o
```

Listing 2: Example makefile.

3 Submission

Git is a version control system. There is a wealth of documentation available on the web, but feel free to visit your friendly neighborhood TA (i.e. Tom) if you are having difficulties. We have set up a server, users, and repositories for use in this course.

Access to your repository is protected via public/private key pairs. Your key and username will be provided via email after our first meeting. Installing the key on Linux and OS X is fairly simple: first save the key as `~/.ssh/hpckey`. Then open up the file `~/.ssh/config` (create it if it does not exist) and add the lines:

```
Host 134.91.11.132 hpcgit
  Port 5555
  User your-user-name
  IdentityFile ~/.ssh/hpckey
```

To clone a repository named 'repo', use the command line:

```
$ git clone gitolite@hpcgit:repo.git
```

This will create a directory named '`repo`', which you should place your source in.

3.1 Your account information

Your username is the first letter of your first name followed by your last name. For example, since my name is 'Tom Fogal', my username is '`tfogal`'. Usernames are always lowercase.

The repository for this assignment is already created. The naming convention is `as0-username`.

3.2 Automation

When you push your code to the server, it will automatically download the head of your `master` branch, compile it, and run a simple test or two. The tests' output will be returned as part of the 'push' command's output. You may push (and therefore test) your code as many times as you would like without penalty, provided it is before the deadline.

These automated tests are not exhaustive: passing them with flying colors does not necessarily mean you will receive 100% of the credit for an assignment. However, failing any automated tests ensures you will receive a 0.

The smart student tests early and often.