Advanced practical **Programming** for Scientists

Thorsten Koch

Zuse Institute Berlin

TU Berlin

16. June 2017

Programming principles by John Romero

(1) No prototypes. Just make the game. Polish as you go.
Don't depend on polish happening later. Always maintain constantly shippable code.

(2) It is incredibly important that your game can always be run by your team.
Bulletproof your engine by providing defaults upon load failure.

(3) Keep your code absolutely simple.
Keep looking at your functions and figure out how you can simplify further.

(4) Great tools help make great games. Spend as much time on tools as possible.

(5) We are our own best testing team and should never allow anyone else to experience bugs or see the game crash. Don't waste others' time. Test thoroughly before checking in your code.

(6) As soon as you see a bug, you fix it. Do not continue on. If you don't fix your bugs your new code will be built on a buggy codebase and ensure an unstable foundation.

(7) Use a superior system than your target.

(8) Write your code for this game only – not for a future game.
You're going to be writing new code later because you'll be smarter.

(9) Encapsulate functionality to ensure design consistency.
This minimizes mistakes and saves design time.

(10) Try to code transparently. Tell your lead and peers exactly how you are going to solve your current task and get feedback and advice. Do not treat game programming like each coder is a black box. The project could go off the rails and cause delays.

(11) Programming is a creative art form based in logic.
Every programmer is different and will code differently. It's the output that matters.

Exercise: Sorting

Input: A file containing at most 27,000 integers in the range 1 ... 27,000.

It is a fatal error condition if any integer occurs twice in the input.

No other data is associated with the integer.

Output: A sorted list in increasing order of the input integers.

Constraints: At most (roughly) one thousand 16-bit words of storage are available in main memory. There is plenty of disk space available.

Exercise 5

Makefile (default (1st) should be build of program)

-LDFLAGS

Doc target

Coverage target

Test target

No magic numbers like 2000000000

```
g++ -std=c++0x -O3 -Wall ex5.cpp -o ex2
```

```
g++ -O3 -c -Wall  ex5.cpp -o ex5 [-std=c++11]
```

```
Traceback (most recent call last):  File "src/ex5/ex5.py", line 4, in <module>
import networkx as nx
ImportError: No module named networkx
```

```
./ex5 /data/vorlesung/SP/world666.gph
Going to parse the file /data/vorlesung/SP/world666.gph.gph
ERROR : Encoutered Problem opening file: No such file or directory

./ex5 /data/vorlesung/SP/world666
Going to parse the file /data/vorlesung/SP/world666.gph
Vertexcount: 666
Edgecount: 221445
Reading edges...
Creating graph...
Compute shortest paths via Dijkstra...
*** Error in `./ex5': munmap_chunk(): invalid pointer: 0x0000000003f3cb70 ***=======
Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7f959c0317e5]/
…
```

```
./ex5 /data/vorlesung/SP/world666.gph
RESULT VERTEX 665
RESULT DIST 17955
```

```
main.c:33:21: error: ‘file’ undeclared (first use in this function)
     FILE *f = fopen(file, "r");
```

```
public Node(int id)
{
  this.id = id;
  this.distance = Integer.MAX_VALUE;
  this.visited = false;
  this.touched = false;
  this.neighbours = new ArrayList<Neighbour>();
}
```

Exercise 5: Graphs

Write a program that:

- reads in a graph from file given in .grp format (see data at<http://www.zib.de/koch/SP/data> for examples) with the filename provided as a command line argument. Note that the graphs have positive edge weights (that are always below 2^{31}).

- computes a longest (with respect to the edge weights) shortest path from any vertex to the vertex with index 1. In case of ties the vertex with smallest index should be taken.

- produces an output with the following syntax:
  RESULT VERTEX <vertex index>
  RESULT DIST <distance of longest shortest path>
  RESULT TIME <cpu time in seconds>

- You may use graph libraries such as boost (for C++) or graph-tool (for Python), but you are not allowed to copy the entire program.

Exercise 5-b: Graphs

Fix whatever is still to fix with your program.

There should be a makfile for generating documentation, coverage, and any checking you found.

If you not have implemented your own shortest path algorithm, please do so.

If you not have used a library algorithm for this so far, please do so.

There should be a command line switch -m1 for your method and –m2 for the library method.

So ex5 –m1 graph666.phd or ex5 –m2 graph666.phd