A practical guide to computer simulation II

Alexander K. Hartmann, University of Göttingen

May 1, 2003

1 Overview

Structure: half lecture, half free training sessions

23.4 Introduction to C: first program, Compile, data types, commands, math. functions, headers functions

30.4 Introduction to C II, structures, pointers, self-defined data types dynamical memory allocation, command line arguments, macros, Make files

7.5 Debugging: gdb, ddd, memory leaks

14.5 Software Engineering, object oriented programming

21.5 Algorithms: O notation, easy/hard problems, Recursion, Divide-and-conquer, Dynamic Programming

28.5 Data structures: Lists, Trees, Heaps, Hashing tables

4.6 Libraries: numerical recipes, LEDA, own libraries

11.6 PAUSE: Pfingsten

18.6 Random numbers: simple random numbers, exponential, Gaussian, Rejection method,

25.6 data evaluation: Plotting, Fitting, resampling/jacknife

2.7 Presentation/Recherche latex, xfig, INSPEC

9.7 NWG Teffen + Umzug

16.7 Umzug

2 Introduction to C

2.1 My first program

Use Text editor for first.c

```
#include <stdio.h>

int main()
{
  printf("my first program");
  return(0);
}
```

Compile

```
cc -o first first.c -Wall
```

Run:

```
first
```

To learn more about the usage of the printing function: type `man 3 printf` in a shell.

2.2 Variables, Commands, Expressions

Variables have to be declared. Names (e.g. for variables) have to start with a letter (a...z,A...Z) and can include (basically) letters, numbers and underscore (_).

```
#include <stdio.h>

int main()
{
  int counter;

  for(counter=0; counter<10; counter=counter+1)
    printf("%d\n", counter);

  return(0);
}
```

Important types are:

- int
- double
- char
- arrays=pointers
- functions
- structures
- self-defined data types
- any combinations

One can define several variables at once, e.g. `double number1, average, sum`. Important commands are

- assignment, arithmetric expressions, e.g.

```
a = b + c;
value = 3.0*sin(angle);
hours = minutes/60;
mod = x % y;
value += delta;
counter++;
number--;
code = input & mask     /* bitwise AND */
code2 = input | mask2   /* bitwise OR */
rest = sequence << n    /* bit shift left by n bits */
```

- for-loop

```
for( <initial> ;  <end-condition> ;  <command>)
  <command>;
```

- while loop

```
while( <condition> )
  <command>;
```

  Example

```
while( counter < 10 )
  printf("counter=%d\n", counter++);
```

- if-statement

```
if ( <condition >)
  <command1>;
else         /* optional */
  <command2>;
```

  Possible conditions

```
a==b   !!!!!!
a!=b
a<b
a<=b
a>b
a>=b
```

  Conditions can be connected by AND (&&) or by OR (||), e.g.

```
if( (money>100) || ((price < 10) && (money>20)))
  printf("Buy it!!\n");
```

- blocks for use instead of one command

```
{
  <command1>;
  <command2>;
  ...
}
```

Attention

```
double x;
x = 2/3;
printf("%f\n", x)
```

prints 0.
Casts translate datatypes if possible:

```
int z;

z = (int) 3.2;
printf("%d\n", z);
```

prints 3;

2.3 Arrays

Arrays = vectors or "Lists" of variables. Declaration: `<type> <name>[<size>]`.
Arrays start at zero an run up to `<size>-1`. Example:

```
#include <stdio.h>

int main()
{
  int counter;
  double value[10];
  for(counter=0; counter<10; counter++)
    value[counter]=counter*counter + 0.1;
  return(0);
}
```

An array is an array of arrays: `double matrix[10][10]`
Sum of two 10x10 matrices

```
for(row=0; row<10; row++)
  for(column=0; column<10; column++)
    result[row][column] = matrix1[row][column] + matrix2[row][column];
```

2.4 Subroutines, Functions

There are predefined functions, e.g. mathematical functions. One needs a header-file, e.g. : `math.h` (strange results if used without)

```
#include <stdio.h>
#include <math.h>

int main()
{
  double argument;

  argument = 0.0;
  while (argument < 2*M_PI)
  {
    printf("sin(%f)=%f\n", argument, sin(argument));
    argument += 0.1;
  }
  return(0);
}
```

Include library when compiling: `-l<lib-name>`

```
cc -o first first.c -Wall -lm
```

Random numbers

```
#include <stdlib.h>

...

value = drand48();
```

generates random number in $[0, 1)$
Define your own functions:

```
#include <stdio.h>

/*************** calc_average()**************/
/** Calculates average of values passed      **/
/** uses periodic boundary conditions        **/
/** PARAMETERS: (*)= return-paramter         **/
/**       number: .. of values               **/
/**     value: array of values               **/
/** RETURNS:                                 **/
/**    average                               **/
/*********************************************/
double calc_average(int number, double value[])
{
  int counter;  /* local variable, not visible e.g. in main() */
  double sum;   /* the same */
  sum = 0.0;
  for(counter=0; counter<number; counter++)
    sum += value[counter];
  return(sum/number);
}

int main()
```

```
{
  int counter;
  double value[10];
  double average;
  for(counter=0; counter<10; counter++)
    value[counter]=counter*counter;
  average = calc_average(10, value);
  printf("avg=%f\n", average);
  return(0);
}
```

Subroutines = functions without return value:

```
void print_value_and_flowers(int x)
{
  printf("flowers\n, value=%d, flowers\n", x);
}
```

More than one return value: use pointers (see later)

2.5 Scope of variables

Variables are local

```
#include <stdio.h>

void change(int z)
{
  int x;
  x= 100;
  printf("local x=%d, z=%d\n", x, z);
  z= 101;
  printf("local x=%d, z=%d\n", x, z);
}

int main()
{
  int x, z;

  x = 200; z =201;     /* one can have several commands in a line */
  printf("x=%d, z=%d\n", x, z);
  change(z);
  printf("x=%d, z=%d\n", x, z);

  return(0);
}
```

2.6 Excercise: ballistic deposition

Simple model for sputter deposition of atoms on surface, here 1 dimensional.

- system of size L. use e.g. `#define L 10` to store size in program.

- particles fall down at randomly chosen columns.
- they stop at the first particle they touch [PICTURE]
- use periodic boundary conditions
- Realization: use array $h[\ldots]$ of size L to store heights, initially 0
- use one function `void depose(int size, int h[])` for one particle falling down
- one sweep: L particles fall down.
- Make 100 sweeps for small systems like 10,20,50.
- Use function `void roughness(int size, int h[])` to calculate roughness $W = \sqrt{\langle h^2 \rangle - \langle h \rangle^2}$. The $\langle \ldots \rangle$ denotes the average over all heights. Record roughness as a function of sweeps.

References

[1] R. Berrendorf, *Einführung in die Programmiersprache C* `http://www.theorie.physik.uni-goettingen.de/support/Rechner/Programmierung/c.ps.gz`

[2] A.K. Hartmann and H. Rieger, *A practical guide to computer simulation* `http://www.theorie.physik.uni-goettingen.de/support/Rechner/Programmierung/simulation_guide.ps.gz`

[3] Brian W. Kernighan and Dennis M. Ritchie, *The C programming language*, (Prentice Hall, Englewood Cliffs N.Y. 1988)