Very basic elements of programming, and, therefore, of Processing

5 December 2009

Disclaimer. This is not an introduction, nor to Processing. It is only for the memories of students in a course at HfK Bremen, winter term 2009/10.

A program is a text. As a text it is finite and static. The text class called program has, however, a peculiar property: A text that is a program can also be read by a machine, the computer. In order to be capable of doing this, the computer must be properly arranged.[1]

A program is a text that describes in a finite way an infinite set of processes. The program constitutes the schema that the individual processes belong to. They are instances of that schema.

Each instance of a schema is determined by an appropriate set of input data.

Data are objects that programs („operations") act upon. It is, indeed, the program's task to transform the input data into output data.

Data can be possessed as values by variables or constants. Variables (and constants) have names, values, and types. The type restricts the set of admitted values. If a variable is of type float, it may take on values of the floating point type only.

Constants are not allowed to change their values. Their names are usually of a conventional kind. E.g., „5" is taken as the name of a constant of type int that has the constant value 5 (in the decimal number system). However, „5.0" is taken as the name of a constant of type float that has the value 5.0, and does not change it.

Data types are sets (of the permitted values) together with permitted operations (at times called methods or functions). Two variables or constants of type int may, e.g., be added. The expression „5 + 6" stands for the value 11 in int-arithmetic.

Expressions are strings of characters built according to a strict syntax. They allow to write, in a program, texts of considerable complexity.

[1] If our program is called P1, there must be another program, call it CL, that treats P1 as data! If our program, P1, is written in the programming language L, then CL must be a compiler or interpreter. It compiles programs written in L into programs written in another language that is closer to the machine's hardware. A computer can read a text as a program only under the control of an appropriate compiler.

winter term 2009/10 | 2 sws | lab workshop
for students of digital media (bachelor, master, registered at HfK or other schools) | guests welcome
extra, free, independent, therefore no credit | independent study possible based on this basic programming course, arranged individually
limited to 20 participants | register by email | bring your laptop | download Processing from www.processing.org
takes place: 21 & 22 nov.; 28 & 29 nov.; 5 & 6 dec. 2009, from 10 to 17 at OAS building, Linzer Str. 9a, Bremen
Frieder Nake | nake@informatik.uni-bremen.de | (0421) 218 3525 Linzer Str. 9a, room 3015 | (0421) 9595 1267 HfK room 2.11.030
office hours Wednesday 12 – 14 at Linzer str. 9a & Tuesday at HfK upon arrangement | www.compart.uni-bremen.de

The Processing language is embedded into a Processing environment, the Processing Development Environment, pde. When you „start" a „sketch" written in Processing, the Processing compiler is executing your program in the pde. This results in a Java program which will then be executed. Or it ends in some error message because of syntactic errors or run-time errors. Usually in such a case, you get an indication of where the error occurred and what the reason is. But don't totally rely on this. You must figure out for yourself.

Each and every variable or function that you want to use in your program and that you don't find in the environment, you must explicitly introduce by „declaring it. You do this by using a declaration (a declarative statement).

Some data types are elementary or primitive, others are structured. Examples of primitive data types are int, float, boolean, char, color (look them up in the references of the Help system). An important structured data type is the array. An array is a finite sequence of elements of one chosen type. The elements of an array can be accessed by so-called indexed names. The index of a name gives the number of the named element in the sequence.

Examples:

```
int i, j, y, x;
// four variables are declared of type int. Their names are i, j, y, and x, resp.

float [] a;
// a variable of the name a is declared to be of the type „one-dimensional sequence
// of floating point numbers"

a = new float[10];
// the variable a (already declared) is now assigned a value at the structure level. It is to be of length 10
for (int h = 0;  h < 10;  h = h+1)  {a[h] = 2*h + 1;}
// the elements of the array  a  are now assigned concrete values, a[0] is 1, a[1] is 3, ..., a[9] is 19.
// These values are stored as float. An int value can be assigned to a float variable. Not the other way.
```

Important statements we have studied are the assignment (=), the iteration (for, while), the alternative (if-then, if-then-else).

It is mandatory that statements end in a semicolon.

Everything in programming is an abstraction. Only those aspects of the real world surrounding us physically, or imagined by us mentally, that we model explicitly, are thereby made recognizable, manageable, computable by a computer, i.e. by a program. Programming thus appears as the art of making things explicit, so explicit, indeed, that even a machine can operate on them.

Any such modeling step is a step of abstraction. We replace what belongs to the world around us, or inside us, by abstractions (and thus reductions). The major feature for such abstractions are names.

Things and relations in a program's world are operands (objects) and operations (functions). Operations are applied to operands in order to change the status of those operands.

Abstracting static elements leads to data types. Abstracting dynamic elements leads to functions. Therefore, another very important feature of programming languages is functional abstraction. The syntax of a programming language must provide means to declare a function and to use it.

```
void myFunc();
{ ..... all the statements needed to describe what myFunc is to do ..... }
// this statement declares a function of the name myFunc.
```

```
// It does not return a value. Therefore, void is its type. It does not have parameters. Therefore, ()
// The function's „body" is still not specified.

myFunc();
// The function myFunc is called, which means, it is executed, an attempt to execute it is made

int yourFunc( int i; float x)
{ ..... all the necessary statements ..... return .... ;}
// a function by the name yourFunc is declared. It has two parameters, i of int type, and x of float type
// This function returns a value of int type. Therefore, the code of the body must end with a return

if (yourFunc(7, 5.1) == k)  { ..... } else  { ..... }
// The function myFunc is executed. It is applied to the constant parameter values 7 and 5.1, the result is
// compared with the value of k. Depending on the result, one of two alternative paths is chosen
```

The most frequently used structure of a Processing program is

```
any declarations of variables needed later

void setup()
{ ......... }

void draw()
{ ......... }

any declarations of your own functions
```

This says that there will usually be two functions that have standard names, i.e. setup und draw, and pde usually expects those to exist. The function setup (with no parameters) gets automatically executed just once. It may change certain parameters in the environment (the size of your output graphics window, e.g.).

The function draw is then executed indefinitely, until a programmed stop occurs. It is executed at the highest frquency possible and close to the value of frameRate (which is 60 by default). The interplay of these conventions is responsible for much of the charm of Processing. When it comes to interactive use of programs, some more features of the function organization must be taken care of.