Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Introduction to Programming

March 20, 2018

Alex Mitrevski

What is Programming?

Loosely speaking, programming means "giving detailed instructions to a computer about the way it should behave"

Detailed instructions because the computer is just a machine

On their own, computers have no intelligence, so we have to tell them what to do

Algorithm

An algorithm is *a set of **detailed** instructions that tell a computer how to behave*

Example algorithm: going to the supermarket

1. Take pen and paper
2. Make a list of things to buy
3. Leave the pen on the table
4. Put the paper in the pocket
5. Go out of the door
6. ...

Program

A program is a set of algorithms synchronized in a way that solves a specific problem

Example program: go to the supermarket

1. Execute algorithm *Prepare*
2. Execute algorithm *Drive to the market*
3. Execute algorithm *Shop in the market*
4. Execute algorithm *Go back home*

Computer Instructions

In everyday life, an instruction can be thought of as a recommendation (or a rule) for doing something in a certain way

In programming, instructions represent:

- Saving values in memory
- Retrieving values from memory
- Performing operations on the values stored in memory

Performing Instructions (2/2)

Most of the time, we programmers work with *computer memory*

A useful analogy for memory is to model it by boxes and blocks (I owe this very useful model to my first programming professor, Dr. Matthieu Puigt)

In real life, a box is used to store isolated objects; in programming, a **variable** is what is used to keep such objects

Performing Instructions (2/2)

In real life, domino pieces are kept together, as a whole block of objects; in programming, we can use **arrays** or **lists** to simulate these blocks

Inside the computer, boxes are **registers** or special **memory addresses**; blocks are **sequences** of consecutive memory addresses

Why Do We Work With Memory

Every communication requires a medium; in programming, memory is the communication medium between programmers and computers

Making high level models of memory (especially visual models) can help you understand your programs

A Classic Problem with Variables (1/4)

Consider the problem of swapping the values of two variables a and b

Suppose that we have the following situation: the variables a and b that store the numbers 5 and 10

Figure 1: Two variables example

A Classic Problem with Variables (2/4)

In programming, a memory address (a box) can store only one value at a time

What happens if we put the value of b in a? We will end up with a situation as follows:

Figure 2: Two variables: b assigned to a

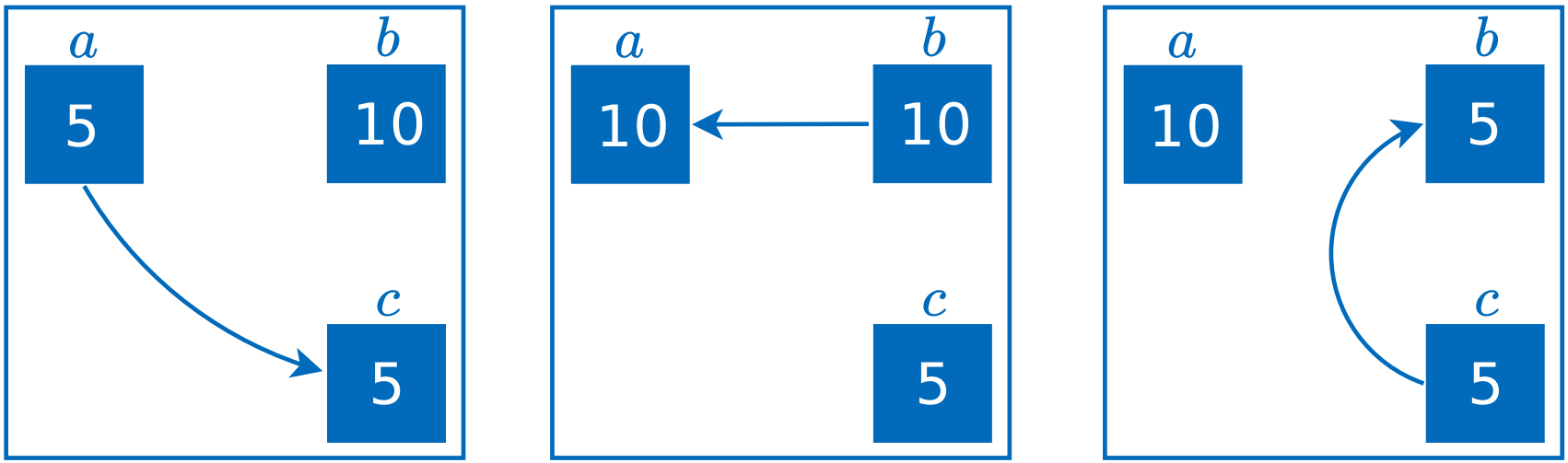
A Classic Problem with Variables (3/4)

If we just assign b to a, we will lose the value of a

How are we actually going to solve the problem of losing the value of a variable?

A Classic Problem with Variables (4/4)

To solve the swapping problem, we are going to introduce a new, temporary variable



Variables Conclusion

The previous example illustrates two essential things about programming:

- The use of variables: everything we do in programming involves manipulating variables in some way
- The necessity for knowing what happens with the variables in your program **at any time**: if you don't understand your program, it's **highly likely** that you will make a mistake

Why Do We Need Decisions?

Computer programs would be useless if they could not make decisions

Example: suppose that you have three web browsers installed on your computer and you make browser 3 the default one. Once you do that, you expect that when you click on a link, a page will open in browser 3

The browser example illustrates a program that makes decisions. In this case, the operating system remembers our choice of default browser, so when we open a link, it checks

- Is browser 1 default? No
- Is browser 2 default? No
- Is browser 3 default? Yes. So open the link in browser 3

How Do We Program Decisions?

In most programming languages, the keywords for telling a program that it should do a decision at some point are **if** and its buddy **else**. If and else have the same meaning as in the English language

Decisions are always made by writing expressions that result in a **Boolean** value (i.e. that are either true or false)

```
a = 1
b = 2
if a > b:
    print('a is greater')
else:
    print('b is greater')
```

How Decisions Change the Flow of a Program

Given a program with an if and else conditions, if the value of the logical expression in the *if* condition is true, the program executes only the statements in the if block

If the value of the logical expression in the *if* condition is false, then the program proceeds by executing the statements in the *else* block

Multiple decision branches (i.e. more than two) can be introduced by either nesting if/else blocks or using **else if** blocks (shortened to *elif* in Python); these two variants are equivalent, but else if blocks make programs easier to read

Repetitive Actions: Loops

Programs that perform only declarations, assignments, and decisions are not everything that we can do

Suppose that we want to sum 10000 numbers. Are we going to declare 10000 variables and sum them? Absolutely not; we can use **loops** for that

Loosely speaking, loops are actions that are performed repetitively

Types of Loops

There are two types of loops in programming languages:

- **for loops**, which run a predetermined number of times (for instance, they are very useful for accessing all array elements)

```
numbers = [1,2,3,4,5,6,7,8,9,10]
for i in xrange(len(numbers)):
    print(numbers[i])
```

- **while loops**, which run as long as some condition is satisfied

```
n = 1
while n < 1e10:
    n = n * 10
    print(n)
```

For Loops Explained

A for loop:

- Starts by assigning a variable (we call it a counter) and checking whether the variable is less than the maximum allowed value
- If it is, the statements inside the for block are executed
- Finally, the counter variable is incremented and the test if its value is less than the maximum allowed value is performed again

Many programming languages (e.g. Python and C++ since C++11) have special types of for loops (foreach loops) that iterate over all elements of a given collections (e.g. a list)

While Loops Explained

While loops generally do the same thing as for loops

The difference between for and while loops is that for loops run a predetermined number of times; while loops, on the other hand, can run a potentially infinite number of times

Note that in while loops you **must not** forget to (somehow) update the value of the variable included in the while condition; otherwise, you will run into a situation called **infinite loop**

Statically and Dynamically Typed Languages

A statically typed language determines the type of a variable at compile time (e.g. C++, Java)

In dynamically typed languages, the type of a variable is determined at runtime (e.g. Python, JavaScript)

Dynamically typed languages are generally easier to write, but harder to debug

Variable Types

The type of a variable (remember, think of it as a box) determines what the variable (the box) can store

In a statically-typed languages, if a variable is of type X, it can **only** store values of type X and nothing else

All programming languages have what we call **primitive types**; these are predefined types that we can use "off the shelf" (e.g. *int*, *float*, *char*)

Number Types

Most languages have more than one type for storing numbers

The number types differ by the range of numbers that they can store, but also by the actual memory that they occupy

Compilers and Interpreters (1/2)

Computers are binary machines (understand only 0s and 1s)

On the other hand, programming languages have phrases close to human languages

Why are the computers able to understand our programs then? Because programs are translated with the help of either compilers or interpreters

Compilers and Interpreters (2/2)

Compilers "translate" a program once the program is written, i.e. when a programmer decides to compile (translate) the program (e.g. C++, Java)

Interpreters translate a program line by line as the program is executed (e.g. Python, JavaScript, Ruby)

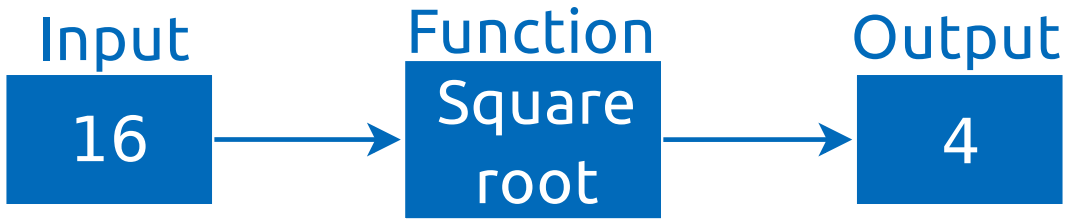
Assembly Language

Programs are not directly translated to machine code, but are first translated to an intermediate language, called **assembly language**

Assembly language is the bridge between higher level languages and low-level machine code

What is a Function?

In mathematics, a function is like a machine that takes an input and gives a specific output. Example: square root



In programming, we have two types of functions:

- Functions that do the same job as mathematical functions (i.e. return a result)
- Functions that do not return a result, but only perform actions

In languages like C++, the **void** keyword in front of the function name tells that the function does not return a value

So, What Exactly is a Function?

A function is a piece of code that does a specific job and can be used with different inputs and/or in many different places in a program

We use functions because they allow us to separate the logic in a program; they also let us (and other programmers) reuse the code

A Function Example (1/3)

Let's suppose that we want to find the maximum of two numbers

In principle, we could explicitly check which number is greater every time we need to do this, but our code would then be very repetitive

A better approach would be to define a function $max(a, b)$, which returns the larger number; we could then call this function anywhere else in our code

A Function Example (2/3)

Here's a Python example of such a function (note: the max function is already predefined in Python, but we define it here for illustrative purposes)

```
def max(m, n):
    if m > n:
        return m
    return n
```

And here's an example of how we would call the function given that we have two variables a and b that have the values 1 and 2 respectively

```
a = 1
b = 2
greater = max(a,b)
```

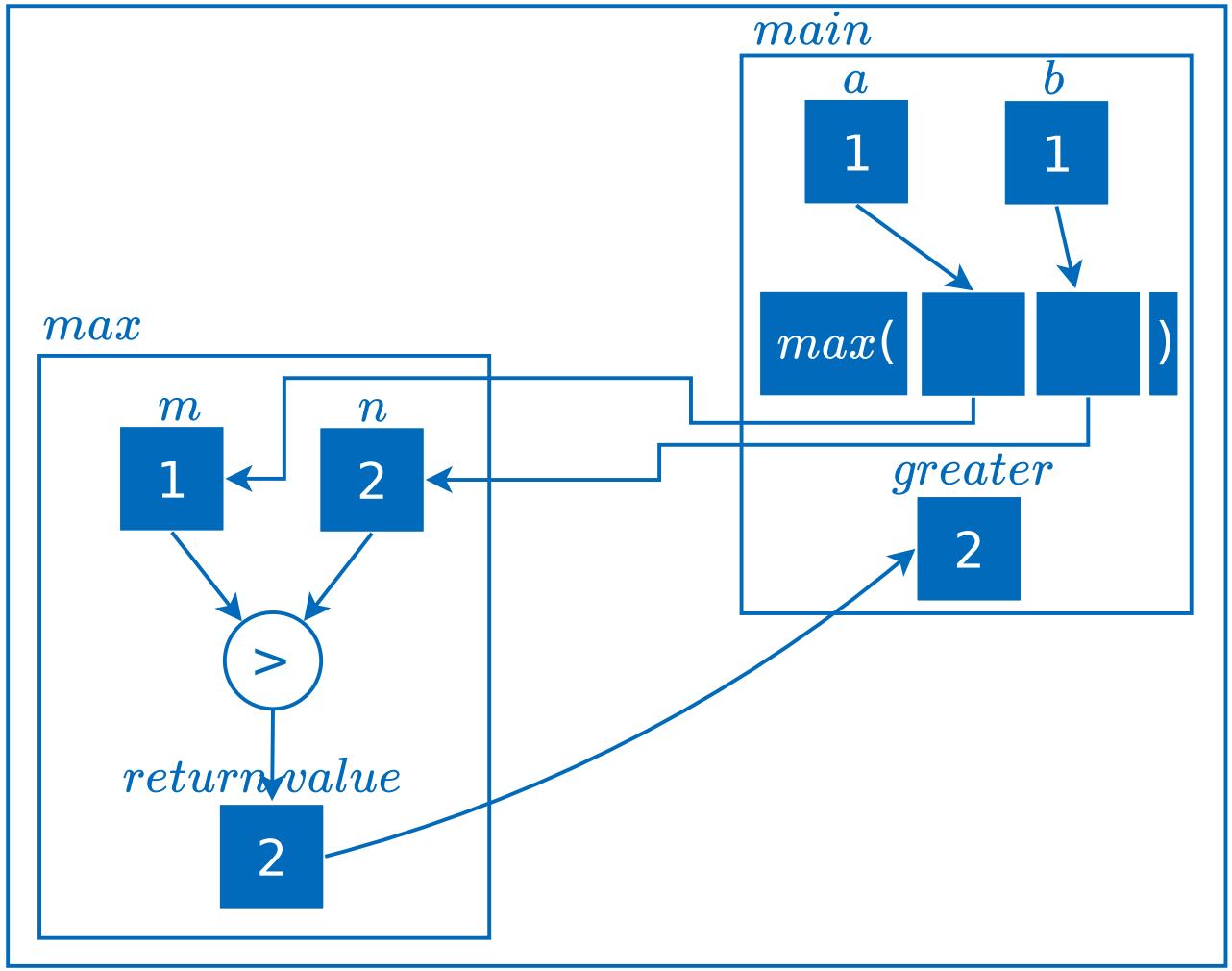
A Function Example (2/3)

In our example, we used a and b for the variables in the main program scope, but m and n in our max function. Why can we do that?

Because the function parameters are just copies of the original values. When we call $max(a, b)$, the values of a and b are **copied to** m and n respectively, so they are not actually the same variables

In some languages (e.g. C++), it is possible to pass a **reference** to a variable; in such case, a and m and also b and n would point to the same address in memory, but they would still be different variables

The Function Example Visualised



Imperative and Procedural Programming

In this slide set, we only talked about two programming paradigms:

- **imperative programming**, which is concerned with the state of a program and the commands that modify that state
- **procedural programming**, which heavily utilises function (procedure) calls

Procedural programs are also imperative programs, but they use functions for controlling a program's state

These are however not the only existing programming paradigms

Object-Oriented Programming (OOP)

OOP is perhaps the most common programming paradigm in even moderately complex projects

OOP-based programs define **classes** from which one can instantiate **objects** that have:

- **fields** that represent the objects' own internal state and
- **methods** that manipulate that state

The OOP principle of **encapsulation** allows hiding part of a program's logic from the users of a particular program module

Languages such as C++ and Python (both of these languages are heavily used by our research group) are not fully OOP-based, but support OOP pretty much completely; on the other hand, languages such as Java and C# are completely OOP-based

Logic Programming

Logic programming is a very different paradigm in which we represent:

- general knowledge about a certain domain as a set of logical rules
- specific knowledge about the domain by asserting facts

A program written in a logic programming language is concerned with proving new facts from the set of known rules and facts

You will briefly look at one logic programming language (Prolog) in the AI course

Functional Programming

Functional programming, like procedural programming, operates by using functions

Functions in functional programming are however closer to mathematical functions, i.e. they cannot change the state of data

Functional programming is more closely examined in the Master of Computer Science program; we don't do much functional programming in MAS

Thanks for your attention