Fundamentals of Programming 1

Introduction

Arkadiusz Chrobot

Department of Information Systems

October 18, 2024

Outline

Contact Information

Lecturer: Arkadiusz Chrobot, PhD
Room number: 3.23, D building
Office hours: Thursday, 12:00 – 14:00
Phone: 41 34–24–185
E-mail: a.chrobot@tu.kielce.pl
www: https://achilles.tu.kielce.pl/portal/Members/84df831b59534bdc88bef09b15e73c99

Literature

Primary Literature

1. **Brian W. Kernighan, Dennis M. Ritchie,"The C Programming Language", Second Edition, Prentice-Hall Inc., Upper Saddle River, 2012**
2. **Stephen Prata, "C Primer Plus", 6th Edition, Addison-Wesley, Upper Saddle River, 2015**
3. Zed A. Shaw, "Learn C the Hard Way: Practical Exercises on the Computational Subjects You Keep Avoiding (Like C)", Addison-Wesley, Upper Saddle River, 2016
4. Paul Deitel, Harvey Deitel, "C How to Program", 8th Edition, Pearson Education Inc., Hoboken, NJ, 2015
5. Jon Bentley, "Programming Pearls", Addison-Wesley, Inc., Upper Saddle River, 2000
6. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman,"Data Structures and Algorithms", Addison-Wesley Inc., Upper Saddle River, 1987

Literature

Advanced Literature

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, "Introduction to Algorithms", 3rd edition, MIT Press, Cambridge US, 2009
2. Donald E. Knuth, "The Art of Programming", Vol. 1 – 4A, Addison-Wesley Inc., Reading, Massachusetts, 2011
3. Robert Sedgewick, Kevin Wayne, "Algorithms", 4th edition, Addison-Wesley Inc., Reading, Massachusetts, 2011
4. Steven S. Skiena, "The Algorithm Design Manual", Springer-Verlang, London 2010
5. Jens Gustedt, "Modern C", Manning, New York, 2020 (online: https://gustedt.gitlabpages.inria.fr/modern-c/)

Literature

Webpages

1. Wikibooks: C Programing
2. The GNU C programming tutorial
3. Learning GNU C
4. The GNU C Library

Programming

Definition of Programming

Programming is a task of preparing a program for a *computer system* that solves a given problem. It consists of the following steps:

1. building a model of the problem,
2. creating *an algorithm*,
3. writing the algorithm in a programing language,
4. removing syntax and logical errors (so called bugs).

Computer Program

Definition of a Computer Program

A computer program is *an algorithm* that solves a specific problem and is expressed (coded) in a programming language.

Algorithm

Definition of an Algorithm

An algorithm is a series of precise, well-defined activities necessary to complete a given task.

Properties of an Algorithm

Finiteness: The algorithm has to end (i.e. give an answer for the problem it solves) after a finite number of steps. Procedures that possesses algorithm's all properties except for finiteness are called *computational methods*.

Definiteness: Each step of the algorithm has to be specified in a strict, accurate and unambiguous way.

Input data: The algorithm has zero or more input data.

Output data: The algorithm yields one or more output data related to the input data.

Effectiveness: The algorithm should not only complete in finite time, but the time should be as short as possible.

Expressing the Algorithm

The algorithm can be expressed in a form understandable to a human or in a form understandable to a computer system.

Expressing the Algorithm

Problem Description

Euclid's Algorithm for finding the Greatest Common Divisor (GCD) of two integer numbers.

The Problem

Given a two integer numbers M and N find their Greatest Common Divisor, i.e. the largest positive integer number, that divides both of them.

Expressing the Algorithm

Pseudocode

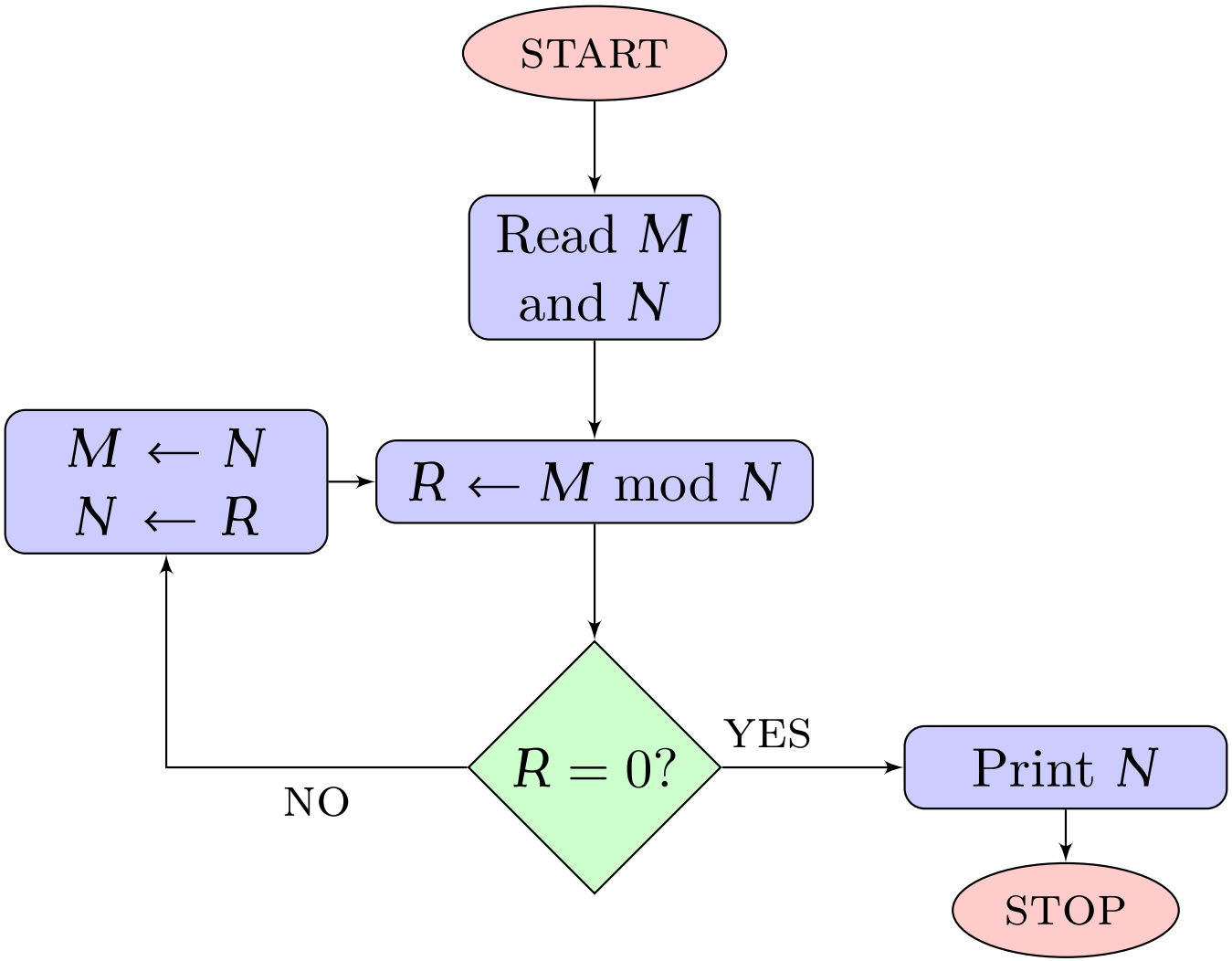
E1.[Finding the reminder] Divide M by N. Denote by R the reminder. $(0 \leq R < N)$.

E2.[Is zero?] If $R = 0$ then stop. N is the answer.

E3.[Reduction] Assign $N \rightarrow M$ and $R \rightarrow N$. Go back to the step E1.

Expressing the Algorithm

Flowchart



Computer System

Definition of a Computer System

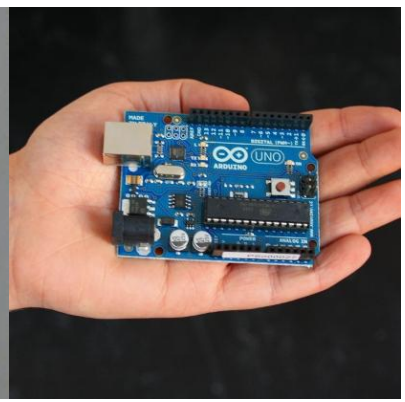
A computer system is a device or a group of collaborating devices that are capable of running a computer program.

Types of Computer Systems

(a) Cluster

(b) Personal Computer

(c) Mobile Device



(d) Microcontroller

...and many, many others ...

Types of Computer Systems – Common Elements

Any computer system has at least two elements:

1. a Central Processing Unit (CPU),
2. a Memory.

Any computer "understands" binary code.

Expressing the Algorithm – a Computer Program

Initially all computer programs (software) were written in machine code, i.e. a series of binary (sometimes octal or hexadecimal) numbers. Next, assembly languages were introduced. In assembly languages each machine instruction is represented by a single mnemonic, i.e. a short, easy to remember string of characters. Latter, a high-level programming languages were developed. A program written in such a language resembles a text in a natural language (usually English). A single instruction in a high-level language can correspond to many machine language instructions. Programs in high-level languages have to be translated to the machine code by a special program called a translator. There are two kinds of translators: interpreters and compilers.

The activity of expressing an algorithm in a computer language is called *implementing* and thus the program is sometimes referred to as *an implementation*. A program written in assembly or high-level language is called a *source code*. It's version translated to the machine code is known as an *executable code*.

Abstraction

The evolution of programming languages is an example of applying an abstraction, which is a method of simplifying a problem by highlighting its most important features and hiding the ones that are unnecessary for solving it. Basically, programming is all about using skilfully the abstraction.

The C Language

Highlights

- developed in 70. of 20th century,
- high-level language with some low-level features,
- supports an imperative, procedural paradigm of programming,
- has a simple syntax that was also applied in many other programming languages (Java, C++, C#, etc.),
- one of the most popular programming languages according to TIOBE rank,
- it is compiled; there are many compilers for many computer systems,
- it is standardised (current standard version is ISO C23, but for the lecture the ISO C99 will be used and a few non-standard GNU C extensions).

The Simplest Program

Listing 1: The simplest program in the C language

```
1  int main(void)
2  {
3          return 0;
4  }
```

The "Hello, World!" Program

Listing 2: "Hello, World!" in the C language

```
1  #include<stdio.h>
2
3  int main(void)
4  {
5          puts("Hello, World!");
6          return 0;
7  }
```

Comments

Comments in a source code are used for explaining the meaning of particular parts of code. They are ignored by a compiler. This feature of comments is sometimes used by the programmers for temporary excluding (commenting out) from the compilation process excerpts of a code that are not yet finished. There are three types of comments available in the C language. Comments of the first type start with the `/*` and end with the `*/` characters. Such comments can have many lines of text. A comment of the second type starts with the `//` characters and ends with the end of line character. Such a comment always has only one line of text. The comments of the last type start with `#if 0` and end with `#endif` preprocessor directives (instructions).

Comments

Examples

Listing 3: Comments in the C language

```
1  /* This is a comment of the first type. Placing other comment of the same type inside
2  this comment is forbidden. */
3
4  /*
5  * It is also a comment of the fist type but in more readable
6  * form.
7  */
8
9  // This is a comment of the second type.
10 // Such a comment may be nested inside comments of
11 // other types.
12
13 #if 0
14 It is a comment of the third type.
15 This comment may have many lines of text.
16 /* It is even possible to place comments of the first type inside
17 comments of the third type. */
18 #endif
```

Cons and Pros of Using Comments

+ they help to understand code,

Cons and Pros of Using Comments

+ they help to understand code,
- they may indicate that the code they explain is not well written,

Cons and Pros of Using Comments

+ they help to understand code,
- they may indicate that the code they explain is not well written,
- changing the commented code makes the comment outdated,

Cons and Pros of Using Comments

+ they help to understand code,
- they may indicate that the code they explain is not well written,
- changing the commented code makes the comment outdated,
+ may be used to temporary “switch off” parts of code that are not yet finished.

Constants

In the programming constants are used for naming values that are time-invariant. There are several ways of defining constants in the C language. The first one involves using so-called preprocessor macros.

Pattern

```
#define NAME VALUE
```

Example

```
#define GRAVITY 9.81
```

Names of constants are usually uppercased. Whenever the compiler (the preprocessors) finds in the source code the GRAVITY it replaces it with the 9.81 value.

Variable

A variable is a name given to a place where data is stored. From the computer point of view the variable is a specific part of its memory. Each variable has three attributes: a name, a scope and a type. The principles of creating a name for a variable are discussed latter in this lecture. The scope determines where it can be used in a program and it is depended on the place in the source code where the variable is declared. We are going to learn about the global variables first. These are available in the whole program, starting from the place where they are declared. The type determines the size of the variable and the sort of information (data) it stores.

Variable Declaration Pattern

Before a variable can be used in a program, it has to be declared first.

```
type_of_variable variable_name;
```

Example

Listing 4: Example of variable declaration

```
1  #include<stdio.h>
2
3  unsigned int number_of_students;
4
5  int main(void)
6  {
7          return 0;
8  }
```

Name of the Variable

Rules

A name is an identifier that allows a programmer to uniformly refer to a variable (or any other element) in a source code. Every identifier (also name of a variable) in the C language must adhere to the following rules:

- identifiers must be unique (there are some exceptions to the rule),
- identifiers must not start with a digit,
- placing special characters which are neither letters nor digits in the identifiers is not allowed, with the sole exception of the underscore character (_),
- the C language is case sensitive,
- identifiers should contain only Latin letters,
- a keyword cannot be used as an identifier (The keyword is a part of the language, for example the `int` word).

Name of the Variable

Recommendations

The rules presented in the previous slide are checked by the compiler. However there are some rules for creating identifiers that are not verified by compiler, but form a convention which helps to make the source code more legible. Below are presented some of them. **Remember, the source code in more often read then written, so it is worth to make it understandable to wider group of programmers.**

- Identifiers should be descriptive.
- Names of variables should contain at least one noun.
- If the identifier contains more than one word, the words should be build of lowercase letters and connected by underscores.
- Single letter identifiers should not be used, except for some specific language constructions (for example: well-known mathematical expressions, loops).

Types of Variables

The Basic (Primitive) Types

Name	Size (in bytes)	Values
`int`	4	integer numbers
`short int` or `short`	2	integer numbers
`long int` or `long`	8	integer numbers
`long long int` or `long long`	8	integer numbers
`char`	1	characters or integer numbers
`float`	4	floating-point numbers
`double`	8	floating-point numbers
`long double`	12	floating-point numbers

One byte equals to 8 bits. Bit is the smallest unit of information that can be processed by a computer (0 or 1). The C language standard does not define the actual size of types of variables, but describes how they relate to each other. The sizes in the table are specific to a 64-bit PC computer.

Binary Number System and Its Derivatives (Basics)

Decimal System

$$128_{(\mathrm{DEC})} = 1 \cdot 10^2 + 2 \cdot 10^1 + 8 \cdot 10^0$$

Binary Number System and Its Derivatives (Basics)

Decimal System

$$128_{(\mathrm{DEC})} = 1 \cdot 10^2 + 2 \cdot 10^1 + 8 \cdot 10^0$$

Binary System

$$1001_{(\mathrm{BIN})} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9_{(\mathrm{DEC})}$$

Binary Number System and Its Derivatives (Basics)

Decimal System

$$128_{(\text{DEC})} = 1 \cdot 10^2 + 2 \cdot 10^1 + 8 \cdot 10^0$$

Binary System

$$1001_{(\text{BIN})} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9_{(\text{DEC})}$$

Two's Complement

$$(-5)_{(\text{DEC})} \Rightarrow \overline{0101}_{(\text{BIN})} \Rightarrow 1010_{(\text{OCB})} + 1 \Rightarrow 1011_{(\text{TCB})}$$

Types of Variables

Ranges of Integer Types for 64-bit Computers

Name	Minimal value	Maximal value
`int`	-2 147 483 648	2 147 483 647
`short int` or `short`	-32 768	32 767
`long int` or `long`	−9 223 372 036 854 775 808	9 223 372 036 854 775 807
`long long int` or `long long`	−9 223 372 036 854 775 808	9 223 372 036 854 775 807
`char`	-128	127

Types of Variables

Characters

Single characters, i.e. letters, digits and non-alphanumerics can be stored in variables of the `char` type. Every value of the variable is interpreted as an ASCII (American Standard Code for Information Interchange) value.

Types of Variables

Specifiers

Specifiers are keywords of the C language that modify the meaning of some types of variables. The specifier `unsigned` is used together with the `int` and `char` types. It informs the compiler that it should interpret the value stored in such a variable as a natural number. In other words it changes an integer number type into a natural number type. Complementary to the `unsigned` is the `signed` specifier, but in the essence it does not do anything and is often omitted. The `long` specifier doubles the size of the variable of `int` or `double` type. The `short` specifier halves the size of a variable of `int` type.

Types of Variables

Ranges of Natural Types for 64-bit Computers

Name	Minimal value	Maximal value
`unsigned int` or `unsigned`	0	4 294 967 295
`unsigned short int` or `unsigned short`	0	65 535
`unsigned long int` or `unsigned long`	0	18 446 744 073 709 551 615
`unsigned long long int` or `unsigned long long`	0	18 446 744 073 709 551 615
`unsigned char`	0	255

In the `limits.h` header file are defined constants for values of limits for every integer and natural number type.

Types of Variables

The `void` Type

The keyword `void` is a data type name, but not a variable type name — all types of variables are data types, but not all data types are types of variables. It means that it is impossible to declare a variable of such a type. However, the keyword is useful in other situations that will be discussed in latter lectures.

Types of Variables

Floating-Point Numbers Types

Some real numbers cannot be accurately represented in the computer memory. For that reason computer scientists created less accurate but possible to fit into the memory representation of these numbers that is called a *floating-point number*. It is based on *scientific notation*, but the base is 2. In the computer memory the numbers can be stored as follows: the most significant bit determines the sign of the number. Some next bits store the significand (sometimes called incorrectly "mantissa"), the last part can be used for storing an exponent. The significand (i.e. the fraction) is expressed in a binary code in such a way, that the each consecutive bit, starting from the left, has a negative base exponent (i.e. $2^{-1}, 2^{-2}, 2^{-3}, \ldots$). The number's exponent is expressed in two's complement. In the C language floating-point numbers are stored in `float`, `double` and `long double` types of variables. The difference between them is not only in the total size of occupied memory but also in the sizes of the parts for significand and the exponent.

Types of Variables

Boolean Type

In the C language all values that are nonzero are interpreted as logically true and all values that are equal zero are interpreted as logically false. In the previous editions of the C language standards there was no definition of a special type of variable for such values. Starting from the ISO C99 standard there is such a type called `bool`. It can be used in a program, provided that the `stdbool.h` header file is included at the beginning of its source code like this: `#include<stdbool.h>`. Variables of such a type can have one of the two values `true` or `false`.

Questions

?

The End

Thank You for Your attention!