Exercises of “Computer Science 1+2”

Jens Schönherr
Fakultät Elektrotechnik
HTW Dresden

March 17, 2025

Preliminaries The programming exercises are to be carried out using Code::Blocks [cod] (cf. Chapter A).

For module testing the tau environment [DPLm] should be used.

1 Introduction

Exercise 1.1 *Convert the given numbers into the given number systems!*

a)

decimal	***binary***	***hexadecimal***	***octal***
10			
20			
25			
35			

b)

binary	***hexadecimal***	***octal***	***decimal***
100			
1101			
10010110			
01110111010010			

c)

hexadecimal	***octal***	***binary***	***decimal***
C			
2D			
FF			
E5			
AFFE			

Exercise 1.2 *Enter the value ranges for the following number representations!*

type	***smallest value***	***largest value***
unsigned 3 Bit		
unsigned 12 Bit		
unsigned 14 Bit		
signed 3 Bit		
signed 12 Bit		
signed 14 Bit		
unsigned 8 Bit		
unsigned 16 Bit		
unsigned 32 Bit		
unsigned 64 Bit		
signed 8 Bit		
signed 16 Bit		
signed 32 Bit		
signed 64 Bit		

Exercise 1.3 *Convert the following number representations into decimal numbers!*

type	***binary***	***decimal***
unsigned	*0101*	
unsigned	*1010*	
unsigned	*1011101*	
signed	*0101*	
signed	*1010*	
signed	*111111*	
signed	*000000*	

Exercise 1.4 *Convert the following numbers into unsigned and/or signed representations!*

decimal number	***unsigned 5 bit***	***unsigned 7 bit***	***signed 5 bit***	***signed 7 bit***
10				
-10				
15				
-16				

2 Programming

2.1 Basic Elements of Programs

Exercise 2.1 *Draw a flow chart for the program from the following Listing:*

```
int main()
{
  int a;

  printf("Input␣a:␣"); // output a message
  scanf("%d", &a);

  printf("a␣equals␣%d␣\n", a);  // output of a as decimal number
}
```

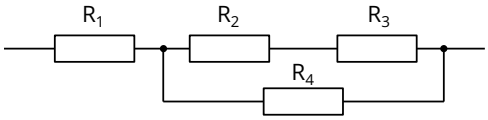
Compile and start the program and check the output for different inputs!

Exercise 2.2 *Develop a C program that converts temperatures from degrees Celsius to degrees Fahrenheit!*

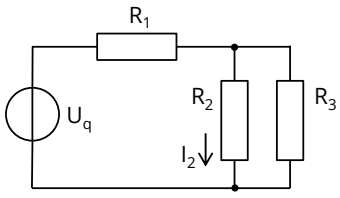
Exercise 2.3 *Develop a C program that determines the new account balance from an account balance (positive) and two deposits! Do not use floating point numbers!*

Exercise 2.4 *Write a C program in which various arithmetic operations are performed with signed and unsigned numbers so that the result is so large or small that it cannot be stored in the specified variable!*

Exercise 2.5 *Develop a C program to determine the total resistance for the following circuit! Start with the development of a flow chart!*



Exercise 2.6 *Develop a C program to calculate the current I_2 in the following circuit! Start with the development of a flow chart!*



Exercise 2.7 *Develop a C program with which the area of a triangle is computed by the length of its sides/edges! Use the Heron's formula! Start with the development of a flow chart!*

Exercise 2.8 *Develop a C program with which the area of a triangle is computed by the coordinates of its vertices in a two-dimensional coordinate system! Use the Gaussian trapezoidal formula! Start with the development of a flow chart! Leave out the computation of the absolute value!*

2.2 Branches

Exercise 2.9 *Develop a C program that calculates the area of a triangle using the coordinates of its vertices in a two-dimensional coordinate system (from Exercise 2.8) so that the absolute value of the area is calculated! Complete the flow chart first!*

Exercise 2.10 *Develop a C program that categorizes a character to be entered (number, upper case letter, lower case letter, special character, control character)! Start with the development of a flow chart!*
Utilize the coding of the characters in ASCII!

Exercise 2.11 *Develop a C program that calculates the heat (in kJ) required for the temperature change of* $1\,\mathrm{kg}$ *water from* ϑ_1 *to* ϑ_2 *(both in °C, with* $\vartheta_1 < \vartheta_2$*)! Start with the development of a flow chart! Use the following assumptions:*

- *specific heat capacity of solid water (ice):* $c_s = 2\,\mathrm{kJ\,kg^{-1}\,K^{-1}}(\frac{\vartheta}{273\,^\circ\mathrm{C}} + 1)$
- *specific heat capacity of liquid water:* $c_l = 4.18\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$
- *specific heat capacity of gaseous water (steam):* $c_g = 2.08\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$
- *specific enthalpy of fusion of water:* $h_{fus} = 332.8\,\mathrm{kJ\,kg^{-1}}$
- *specific enthalpy of vaporization of water:* $h_{vap} = 2256\,\mathrm{kJ\,kg^{-1}}$

Note: It is recommended to develop the program in 3 stages:

1. *restriction to* $\vartheta_1 > 100\,^\circ\mathrm{C}$
2. *restriction to* $\vartheta_1 > 0\,^\circ\mathrm{C}$
3. *no restriction for* ϑ_1

Exercise 2.12 *Set up a truth table for the following Boolean formulas!*

a) $q = (a \wedge \overline{b}) \vee (\overline{b} \vee \overline{a})$

b) $q = \overline{(a \wedge b)} \vee \overline{(\overline{a} \vee \overline{b})}$

c) $q = ((a \vee b) \wedge \overline{c}) \vee (b \wedge \overline{a} \wedge c)$

d) $q = (\overline{\overline{a \oplus b} \wedge \overline{a \oplus c}}) \wedge ((\overline{a} \wedge c) \vee (a \wedge \overline{b}))$

Exercise 2.13 *Develop a C program to calculate the number of days in a month! To do this, the program should evaluate the month as a number (January = 1) and the year. The output should also contain the name of the month. Start with the development of a flow chart! Note the leap years!*

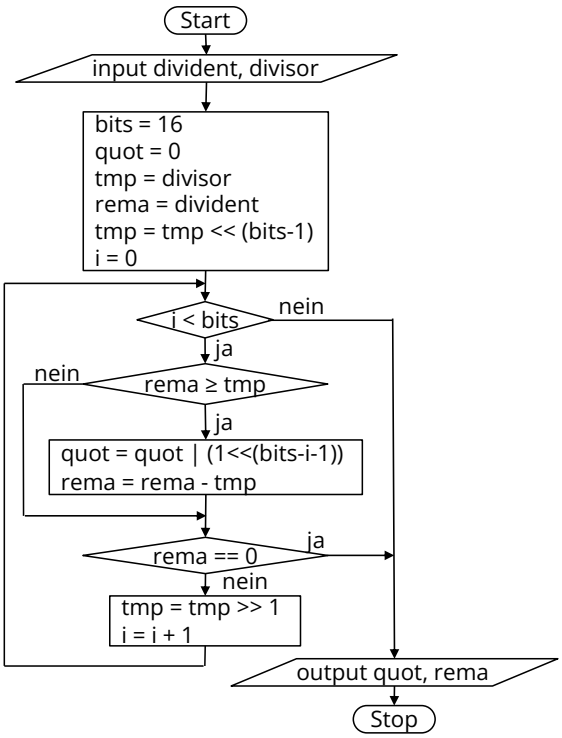
2.3 Bit Operations

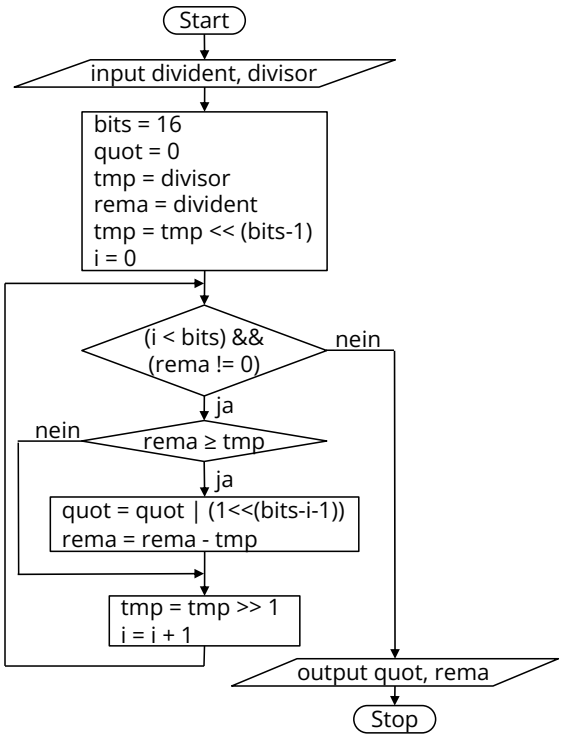
2.4 Loops

Exercise 2.14 *Develop a C program that outputs a table of all powers of two from 2^{-32} to 2^{32}! Start with the development of a flow chart!*

Exercise 2.15 *Develop a C program that outputs all prime numbers up to 65535 and the number of these prime numbers! Start by developing a flow chart! A prime number is characterized by the fact that it is only divisible by 1 and itself.*

Exercise 2.16 *Develop a C program that performs the division of two natural numbers according to the following flow chart! The flow chart shown on the left is to be implemented using a for loop and the flow chart shown on the right is to be implemented using a while loop!*





Variable	***Type***
divident	`uint16_t`
divisor	`uint16_t`
quot	`uint16_t`
rema	`uint16_t`
tmp	`uint32_t`

2.5 Functions

Exercise 2.17

a) *Break down the functionality of the programs developed in Exercises 2.5, 2.6, 2.8, 2.9, 2.10, 2.11, 2.14, 2.15, and 2.16 into a main program for input and output and a function that performs the actual calculation!*

b) *Move the code for the function to an extra module (one module for each exercise)!*

c) *Write tests for each function using tau!*

2.6 Arrays

Exercise 2.18 *Develop a C function that calculates all prime numbers up to 1000 according to the "Sieve of Eratosthenes"! The result should be returned in a field and output in the main program. Avoid using a square root function!*

Exercise 2.19 *In a C program, the following definition can be found within a function:*

```
char s[10] = "abcdefghij";
```

Determine the length of the field and the character string and output the character string! What changes if the string is initialized with `"abcdefghi"` *instead? Give reasons for similarities and differences!*

2.7 Structures

Exercise 2.20 *Write a C program to manage books! Declare a struct for books (title and author) and an array of these book structs! The program shall support adding and deletion of books with title and author, where both entries consist of ASCII characters only. The program should provide user interaction through one-letter commands.*

The program shall contain a module `library` *with the following interface* `library.h`*:*

```
#include <stdint.h>
#include <stdbool.h>

// return success
// reasons for non-success
//   max. number of books already in library
//   title or author too long
bool add_book(char title[], char author[]);

// return success
bool delete_book(uint16_t idx);

uint16_t get_num_books();

// return NULL if idx is invalid
char *get_book_title(uint16_t idx);

// return NULL if idx is invalid
char *get_book_author(uint16_t idx);
```

Write tests for these functions!

Note: For learning purposes, it is useful to first make restrictions at points where it is unclear what a general solution should be and to find a solution under these restrictions. A more general solution should be implemented afterwards.

2.8 Floating Point and Fixed Point Numbers

Exercise 2.21 *Represent the following decimal fractions as floating point numbers according to IEEE 754 (32 bit with 23 bit significand and bias of 127)!*

Decimal	***Significand (bin.)***	***Exponent (bin.)***	***Content of Memory (hexadecimal)***
1			
20			
-0.25			
$6,25 \cdot 10^{-2}$			

Exercise 2.22 *Which decimal numbers are represented by these floating point numbers according to IEEE 754?*

Content of Memory (hexadecimal)	***Significand (dec.)***	***Exponent (dec.)***	***Decimal***
`41580000`			
`c0e00000`			
`bf500000`			
`00000000`			
`7f800001`			

Exercise 2.23 *Represent the following decimal fractions as fixed-point numbers in the format Q11.4!*

Decimal	***Content of Memory (decimal and hexadecimal)***
1	
20	
-0.25	
1.8125	

Exercise 2.24 *Which numbers are represented by these fixed-point numbers in Q7.8 format?*

Content of Memory	***Decimal***
`0x0000`	
`0x7ff0`	
`0xffff`	
`0xfee6`	

Exercise 2.25 *Write a C function which calculated the square root for non-negative numbers of type* `double` *using the interval bisection method! Write tests for this function using tau!*

Exercise 2.26 *Write a C program in which arithmetic operations are performed with floating point numbers of type* `double` *that do not have a number as a result (NaN, $\pm\infty$)! How are these values represented by* `printf()`*? In addition, the program should react to the special cases mentioned with suitable output.*

2.9 Type Casting

Exercise 2.27 *At which point in the conversion of integer operands (Table 2.1) is it possible to falsify the values?*

Table 2.1: Determining the type of an integer operation, (*) `signed long` only if `signed long` can represent all values of `unsigned int`

	`unsigned long`	`signed long`	`unsigned int`	`signed int`
`unsigned long`	`unsigned long`	`unsigned long`	`unsigned long`	`unsigned long`
`signed long`	`unsigned long`	`signed long`	`un/signed long` (*)	`signed long`
`unsigned int`	`unsigned long`	`un/signed long` (*)	`unsigned int`	`unsigned int`
`signed int`	`unsigned long`	`signed long`	`unsigned int`	`signed int`

Exercise 2.28 *Develop a C function that demonstrates the effect of implicit type conversion for integer and floating point types! (if possible: type alignment for operands)*

2.10 Variables

2.11 Enumeration Types

Exercise 2.29 *Write a C program in which the four traffic light phases are run through one after the other (without time delay)! Develop a function that switches from the current traffic light phase to the next one! Use an enumeration type to display the traffic light phases!*

2.12 Dynamic Memory Management

Exercise 2.30 *Change the program for the administration of books (Exercise 2.20) so that the number of books and the length of titles and authors is arbitrary!*

Exercise 2.31 *Change the program for the administration of books (Exercise 2.30) so that the books are stored in a linked list!*

2.13 Input and Output with Files

Exercise 2.32 *Change the program for administration books (Exercise 2.30) so that the datasets are saved to a text file when exiting the program and read from this file when starting!*

Hints/Recommendations:

- *file structure: For each book one line for title and one line for author.*
- *Write the entries with* `fputs()`.
- *Start the read function with as little error handling code as possible and add it at the end.*
- *Read the entries with* `fgets()`. *For this, the length of an entry (title or author) should be limited first.*
- *In a further expansion stage, the read function can be extended so that arbitrarily long lines can be read (multiple calls to* `fgets()` *and concatenation of the strings).*

2.14 Preprocessor

2.15 Complexity

Exercise 2.33 *Specify the complexity class for the following function!*

```
for (int i = 0; i < n; ++i)
{
  for (int j = 0; j < n; ++j)
  {
    double x = 0.0;
    for (int k = 0; k < n; ++k)
    {
      x += a[i][k] * b[k][j];
    }
    c[i][j] = x;
  }
}
```

A Code::Blocks

The following notes refer to Code::Blocks [cod] version 20.03.

A.1 Installation

Code::Blocks is just a development environment without a compiler. On Linux, the GNU compiler is part of the distribution and the corresponding packages may still need to be installed. The Windows installation does not include a compiler. Therefore there is a Code::Blocks version for Windows which includes a GNU compiler. The installation file has "mingw" in the name, e.g. `codeblock-20.03mingw-setup.exe`. If the GNU compiler is already installed on Windows, it may also be used by Code::Blocks.

Under *Settings* → *Compiler* the language standard should be set to C17 and C++17 (Figure A.1 and Figure A.2).

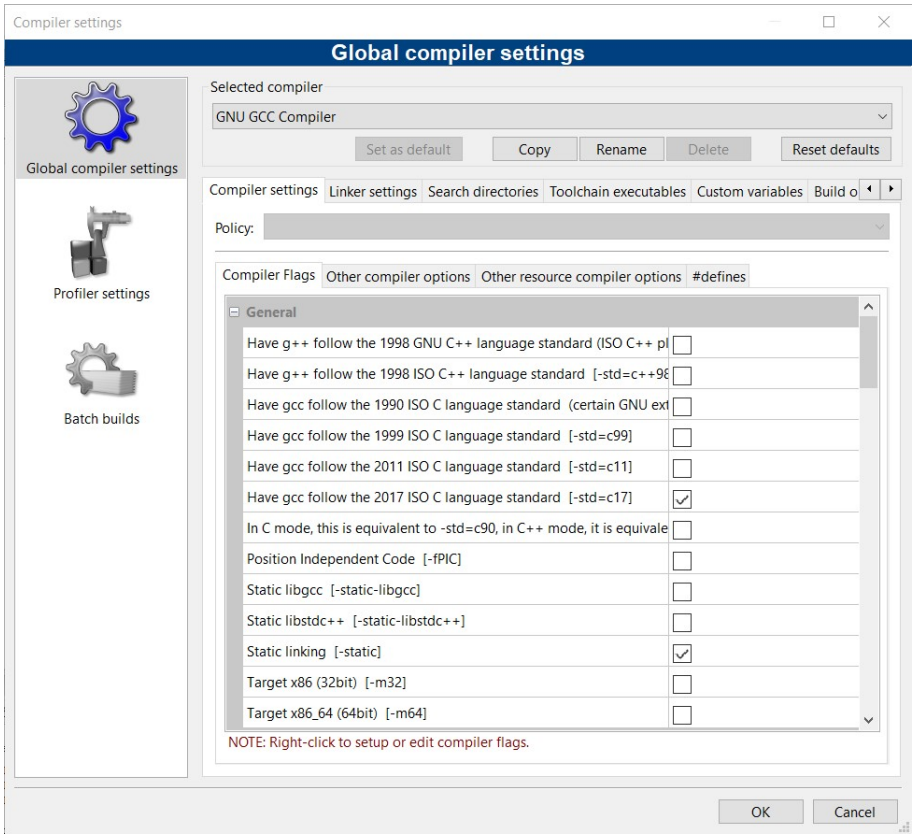


Figure A.1: Compiler-Settings for C in Code::Blocks

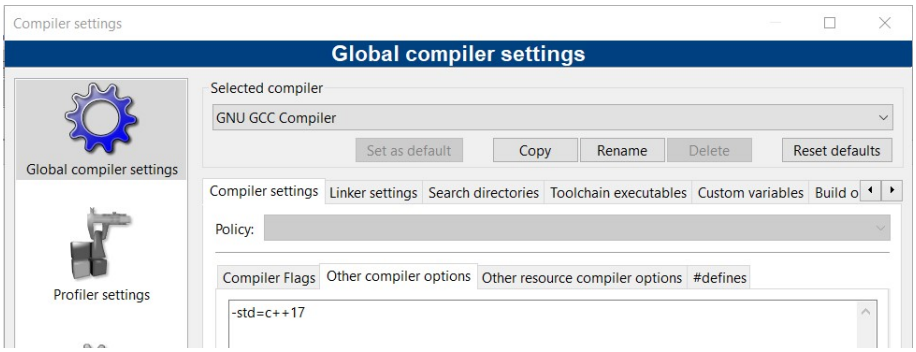


Figure A.2: Compiler-Settings for C++ in Code::Blocks

A.2 Projects

If a project consists of only one C-file, a new file can be opened under *File* → *New* → *Empty file*. This should be saved before writing the code: *File* → *Save file*, select the file type *C/C++ files* and name the file accordingly. The extension `.c` or `.cpp` should be set accordingly if the default setting did not do this correctly.

- The compiler is called via *Build* → *Build* (gear symbol).
- The created program is started via *Build* → *Run* (green arrow icon).
- If both are to be done directly one after the other, *Build* → *Build and run* is available.

The object file (extension `.o`) and the executable file (Windows extension .exe) are created in the same directory.

If a project is to consist of several C/C++ files, a separate Code::Blocks project is created for it via *File* → *New* → *Project* (Figure A.3, *Empty Project* and confirmation with *Go*). It is recommended to create Code::Blocks projects even for projects with just one C/C++ file.

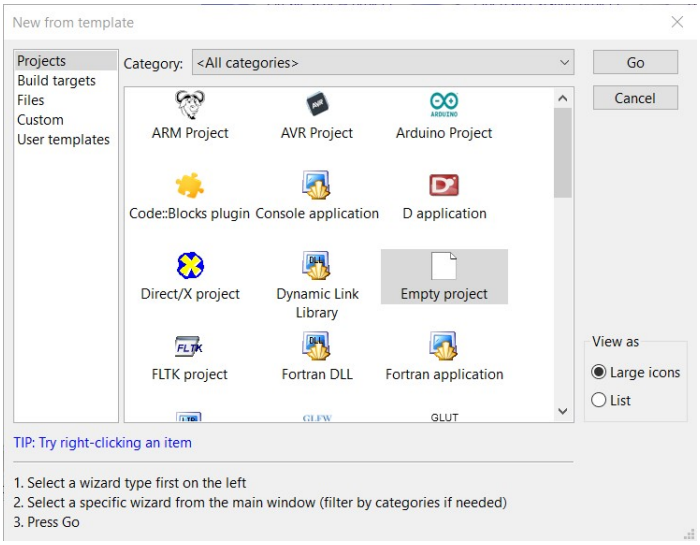


Figure A.3: Creation of a new project Code::Blocks

After that the title and the project directory must be set (Figure A.4). The project directory has the name of the project title and is created in the directory selected in *Folder to create project in:*. If the project directory already exists, it will be used unchanged. The *Project filename:* and *Resulting filename:* fields are automatically filled in and should not be changed.

A file with the project title and the extension `.cbp` is created in the project directory. Project settings are stored in the cbp file. Depending on the project, additional files can be created by Code::Blocks, which have different extensions. All these files use the project title as name.

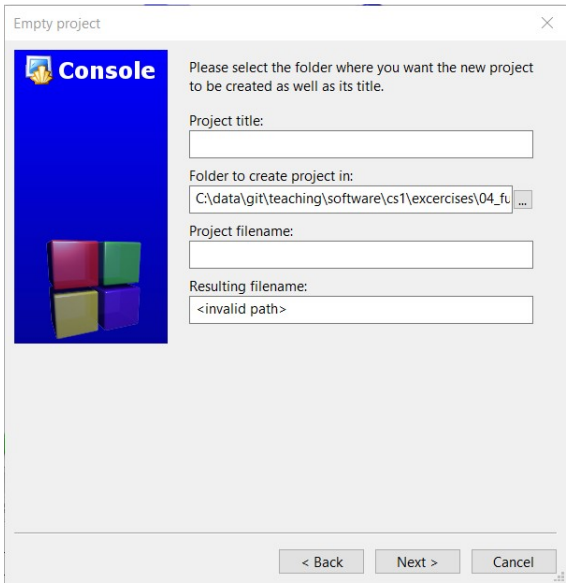


Figure A.4: Creation of a project in Code::Blocks

Creating new files works as described above. However, the file must immediately be given a name and be saved. The question about the assignment of the file to the project must be answered positively. The creation of header files is done in exactly the same way. However, the file extension must be changed to `.h`.

An already existing file is added to a project via *Project* → *Add files...*.

If header files are used in the project which are not located in the project directory, the directories in which the header files are located must be made known. This is done by setting *Search directories* for the *Compiler* under *Project* → *Build options...*. Each path is selected via *Add* in the file system.

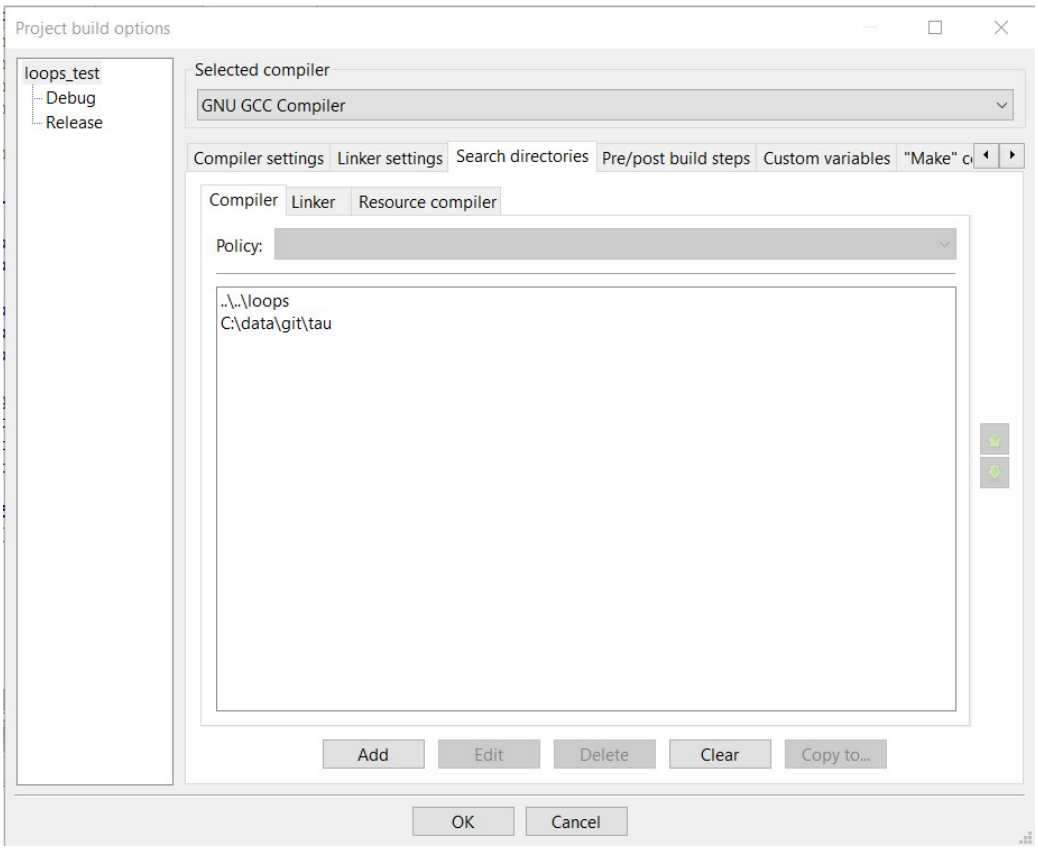


Figure A.5: Setup of search paths for header files in Code::Blocks

A.3 Debugger

To prepare for debugger use on a PC, the setting under *Debug* → *Active Debuggers* should be set to *GDB/CDB debugger: Default*.

To start a debugging session, *Debug* → *Start / Continue* is used. This starts the debugger with the debuggee. If no breakpoint is set, the program (the debuggee) runs without interruption, if necessary until the end. It is therefore advisable to set a breakpoint e.g. to `main()` before starting the debugger. A breakpoint can be set by right-clicking before the actual program line near the line number to open a small menu and selecting *Add Breakpoint*.

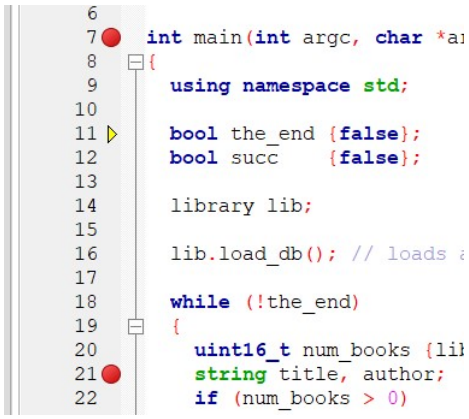


Figure A.6: Source code with two breakpoint (red circles) in Code::Blocks. The programm is currently stopped before line 11 is executed (yellow arrow/triangle).

If the debugger has stopped a program (e.g. at a breakpoint), a yellow arrow is displayed in front of the corresponding line. There are then various options for continuing (for selection, see Figure A.7):

Next line Process the current line (yellow arrow) and stop before processing the next line.

Step into If the current line does not contain a function call, then the behavior corresponds to *Next line*. Otherwise, the program jumps to the function and stops before the first line of the function.

Step out The current function is processed to the end and the programm returns to the caller function. The program stops at the line after the current function has been called.

Continue Program execution is continued until the next breakpoint or until the end of the program.

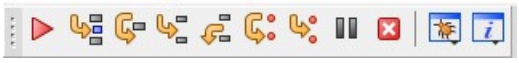


Figure A.7: Toolbar for debug functions in Code::Blocks; if the mouse is held over an icon, its function name appears

.

The values of the variables of the current function can be displayed in the "'Watches"' window. This can be opened in the debugger toolbar Figure A.7 via "'Debugging Windows"' (icon with orange-colored bug).

The values of variables can also be displayed via commands (after *Command*, see Figure A.8). The command `print` followed by a variable name can be used to display the current value of the variable. Arithmetic expressions are also permitted after `print`. This also makes it possible to change a variable. All details about the gdb Debugger can be found in [gdb23].

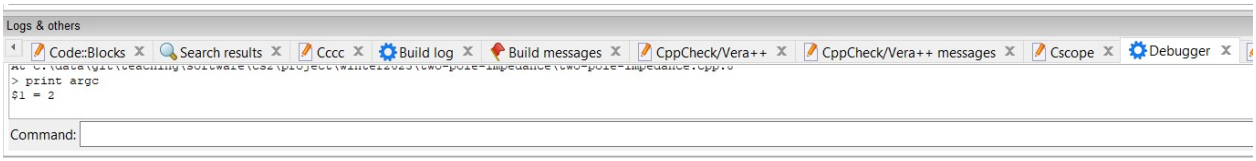


Figure A.8: Input line for debug commands in Code::Blocks, above: the corresponding output; The tab *Debugger* must be selected.

.

Bibliography

[BL80] Buneman, Peter ; Levy, Leon: The towers of Hanoi problem. In: *Information Processing Letters* 10 (1980), Nr. 4, 243-244. https://doi.org/10.1016/0020-0190(80)90150-7. – ISSN 0020–0190

[cod] *Code::Blocks*. Internet. http://www.codeblocks.org/

[DPLm] Dsouza, Jason ; Pidutti, Stefano ; Lisowski, Tomasz ; matu3ba: *tau - A Micro Unit Test Framework for C/C++*. Internet. https://github.com/jasmcaus/tau/

[gdb23] Free Software Foundation: *GDB: The GNU Project Debugger*. online. https://www.sourceware.org/gdb/documentation/. Version: 2023