Introduction to C

Prof. Dr. Peter Braun

7. November 2024

1 Titlepage

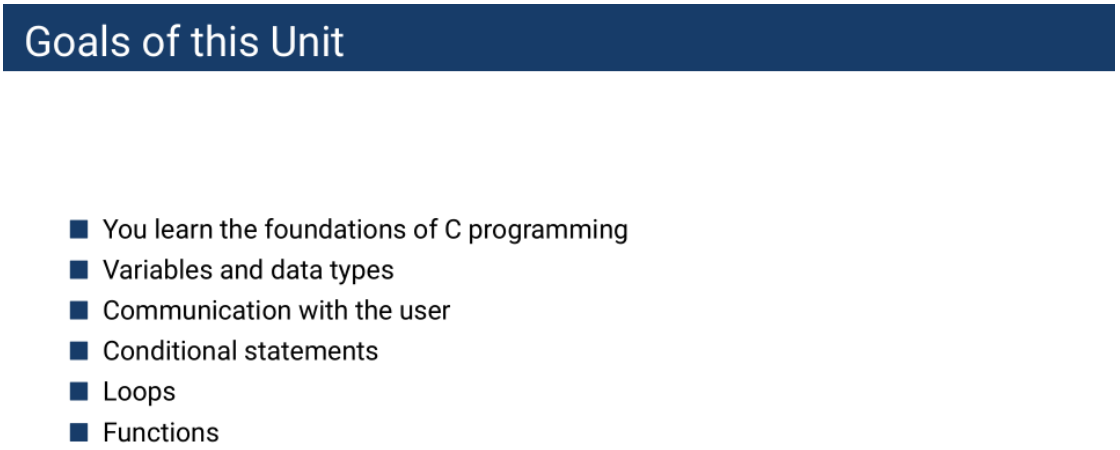
Introduction to C

Prof. Dr. Peter Braun

thws Technical University of Applied Sciences Würzburg-Schweinfurt

No Text – maybe some music :-)

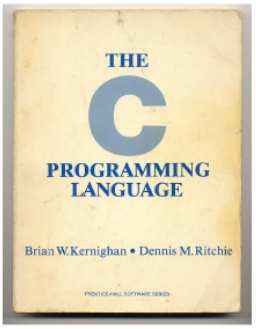
2 Goals of this Unit



Welcome to this first unit on programming in C. In this unit, we will first briefly explain why it is important for every computer science student to learn the C programming language. We assume that you are already familiar with another programming language, for example, Java, and we will also explain the differences between Java and C in our examples. The introduction to programming in C extends over several units. In this unit, we will concentrate on the basics. We will talk about variables and data types. You will learn about functions for outputting data to the screen and reading data from the keyboard. We will also cover common control structures such as branching and looping. Finally, we will also learn about the concept of functions. In the following units, we will look at specific C programming topics, especially memory management and the more complicated concept of pointers.

3 Evolution of C

Evolution of C

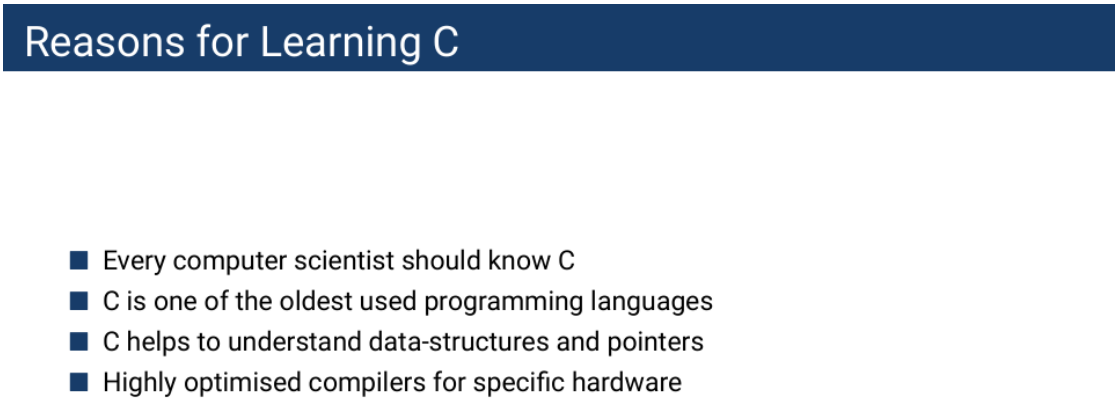


Dennis Ritchie, Ken Thomspon, Brian Kernighan[1]

[1]Left: https://www.bell-labs.com/usr/dmr/www/picture.html
Right: Author unknown, licensed under CC BY

The history of the C programming language and the Unix operating system are very closely linked. Both were developed towards the end of the 1960s and the beginning of the 1970s. The C programming language was designed explicitly to develop the Unix operating system. The very first versions of Unix were written in Assembler. But, as you may have already experienced, software development in Assembler could be more fun. The source code is hard to read and hard to write. And above all, it takes work to port the software to other processors. So, the search was for a programming language that would make software development easier and, above all, portable. But in the end, the software should run just as fast as if it had been written directly in assembler. C is sometimes also referred to as a prettified assembler. The language is very close to the hardware because you can access the computer's memory directly and, therefore, make many mistakes, which can lead to the program crashing. In contrast, C offers some concepts that make the life of a software developer much easier. Although C is over 50 years old, it is still one of the most popular programming languages. Many software systems worldwide, not just operating systems, were written in the C programming language. Alongside Java and other popular languages such as Python or JavaScript, the C programming language still occupies top positions in various rankings on the prevalence of programming languages. Denis Ritchie is considered to be the main developer of the C programming language. Together with Brian Kernighan, he also wrote the first book on the C language. This book is still worth reading today, and you will find the reference at the end of this unit. The third person mentioned in the list next to me is Ken Thomspon. He was one of the developers of Unix and the author of the first version of Unix in Assembler. In the picture, Ken Thompson is sitting in front of a Digital Equipment PDP 11. Dennis Richie is standing next to him.

4 Reasons for Learning C



Every computer science student should have learned the basics of C programming. As mentioned, many software systems are still written in this language today. The probability that you will come into contact with this programming language at some point in your career is quite high. Moreover, it is the forerunner of almost all of today's programming languages. If you know the syntax of C, you will recognize many language constructs in today's programming languages. As mentioned, the C programming language is very close to the hardware. Therefore, Programming in C also helps you understand how a computer or operating system works. Today's programming languages, such as Java, no longer allow us as software developers to get that close to the hardware. They relieve us of many tasks, such as memory management. Once you have experienced the complexity of memory management in this short introduction, you can better appreciate this advantage of modern languages. And last but not least. Programs written in the C language are extremely fast. Today's compilers for C generate highly optimized machine code, and you will hardly find another language with which you can write such high-performance software.

5 Bias against C

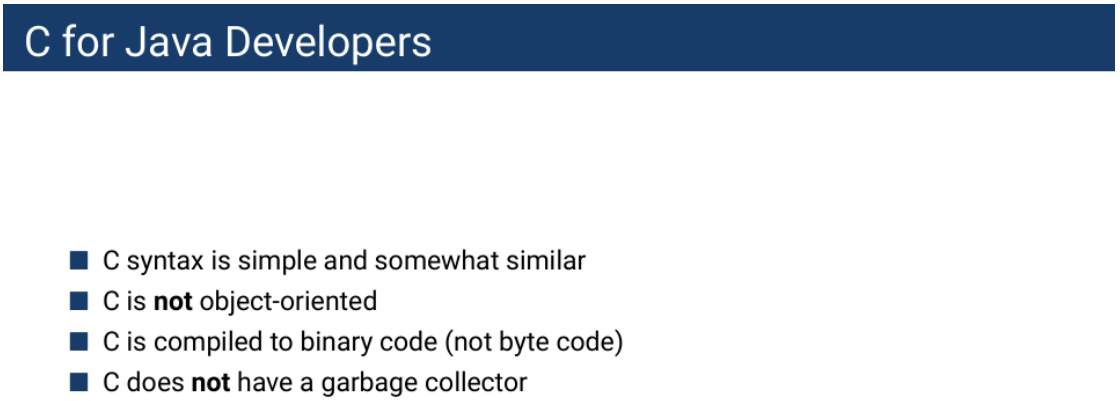
Bias against C

- C is a step backwards compared to Java
- C is only a better assembler
- C programming is dangerous
- C programs are unreadable

```
1  int v,i,j,k,l,s,a[99];
2  main ()
3  {
4    for(scanf("%d",&s);*a-s;v=a[j*=v]-a[i],k=i<s,
5    j+=(v=j<s&&(!k&&!printf(2+"\n\n%c"-(!l<<!j),
6    " #Q"[l^v?(l^j)&1:2])&&++l||a[i]<s&&v&&v-i+j&&v+i-j))
7    &&!(l%=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i]);
8  }
```

Of course, there are also some prejudices against the C programming language. It is an imperative programming language strictly based on the von Neumann architecture. With C, you write software to tell the processor what to do step by step. There are no concepts of object orientation. Even if there are functions in C, it is rare to program in a function-oriented way, as you might know from LISP or Scheme. With C, for example, it is possible to write directly into the memory and thus change states. Side effects are a common problem and a major risk of errors and program crashes. C programs are also usually difficult to read. This is partly due to the compact syntax and the urge of developers to write very short and seemingly high-performance code. Very few programming languages make it impossible to write readable code. But C makes it very easy for the developer. Please take a quick look at the example next to me. We can all agree that it is unreadable. Nobody wants to look for an error in this source code. By the way, the program is a frequently used example in books about C, and it solves the 8-queens problem. All possible positions of the eight queens are displayed on the screen.

6 C for Java Developers



Before starting with the first program in C, we assume you know a programming language such as Java. As mentioned, C is somewhat similar in syntax but much simpler regarding language scope. C has only 32 keywords and is relatively easy to understand. But C is not object-oriented, so there is no concept of classes and objects. Data structures and the functions that work on them are strictly separated. Of course, C is translated by a compiler, but directly into binary code that the processor can execute immediately. Java is first translated into a bytecode, which is then interpreted. Apart from optimization here. However, the biggest difference is that you have never had to worry about memory management as a Java programmer. If you program in C, you must spend much time and effort on this topic. This is because C needs a garbage collector to take over any tasks. You must request memory from the operating system and then release it again. If you do not do this, you will risk a program crash at some point due to a lack of memory.

7 Exercise 1: Hello World

Exercise 1: Hello World

```
1  #include <stdio.h> //Preprocessor commands
2  int main() //main function with return value
3  {
4     printf("Hello World\n"); //strings are null terminated,'\0'
5     return 0;
6  }
```

- Compile the program with `gcc`
- Use the command `gcc hello.c -o hello`
- Run the program with `./hello`

This is a little unusual, as we have yet to show him anything about the C programming language. But let's get straight into the first exercise. Please take a look at the example next to me. We will briefly explain the concepts, and then your first exercise will be to run this program on your computer. Our first example, "Hello World," in C. The only task of this program is to display the character string "Hello World" on the screen. In line one, we have a so-called "include" directive. This practically loads another piece of source code. With such "include" directives, we can better structure the source code and make it reusable. The file we are loading here is called Standard IO point h. The H is the usual suffix for "include" files. The angle brackets give the compiler an indication of where it can find this file. We then define the function with the name MAIN, i.e., the main function of a program. Every C program must have such a MAIN function. The return type must be integer so that you can define the program's EXIT code in the function itself. You could also specify in the round brackets that this program expects arguments. However, we do not consider this case here in the first example. The curly brackets then contain instructions for the main function. In line four, you can see a print statement that displays the text "Hello world" on the screen. In line five, you can see the return statement with which you define the Exit code. The value zero means the program has ended correctly, and no error has occurred. Now open an editor on your computer and type this short program. Then, start the compiler. If you are working on the command line, we recommend using the GNU C Compiler, abbreviated GCC. You can see how to start the compiler next to me on the second point. You will then find the executable program under the name Hello in the working directory. Finally, call up the program. And now, let's take another look at it live on the computer.

8 Screencast

- todo

So, let's work on this exercise together. First we have to check if GCC is available on your system. And you do this by calling the which command as usual. If you see a path here, then GCC is available. And otherwise you have to install GCC first. And you can do this by calling sudo with the apt or aptget install GCC command. And then it takes some seconds and at the end you have GCC available. And now we start with the first C program. We open the editor with main.c, that's the file name. And we just copy what was written on the slide. We start by including standard io.h. We have a main function here that returns an integer. We call function printf with hello world as parameter. Don't forget the carriage return here. We return zero as exit code. We have to save this and we go back to the command line. First we start the compiler GCC and the output should be called hello. You can see here we have a hello program and this already has the correct permissions. It's executable. And so we can call this program and you can see the output here.

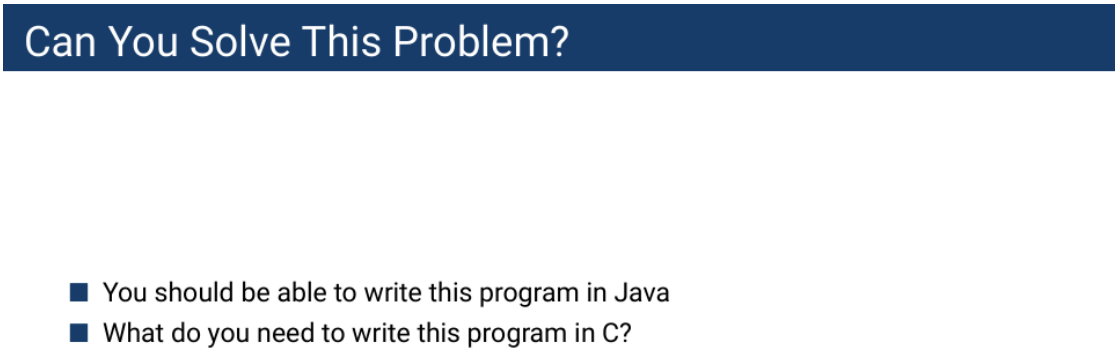
9 Problem: Fibonacci Numbers

Problem: Fibonacci Numbers

- Calculate the Fibonacci Number `n` using:
- `Fib(n) = Fib(n-1) + Fib(n-2)`
- `Fib(0) = 0; Fib(1) = 1`
- The number `n` should be retrieved from user input
- Calculate and print the result

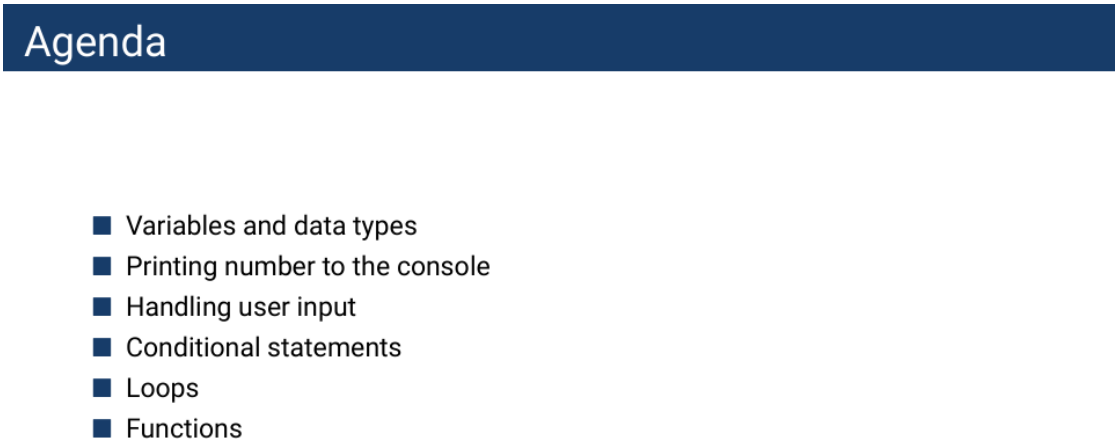
We want to organize our introduction to the C programming language differently than you might have expected. We assume that you already know the basics of programming and have hopefully already programmed simple problems yourself in Java. We will now show you a smaller problem: calculating the so-called Fibonacci numbers. The nth Fibonacci number is the sum of the two previous Fibonacci numbers. As a starting value, we define the zeroth Fibonacci number as corresponding to the number 0 and the first Fibonacci number as corresponding to the number 1. You are now asked to write a program to calculate the n-th Fibonacci number, whereby the number n is to be entered by the user. The computed number should then be displayed on the screen.

10 Can You Solve This Problem?



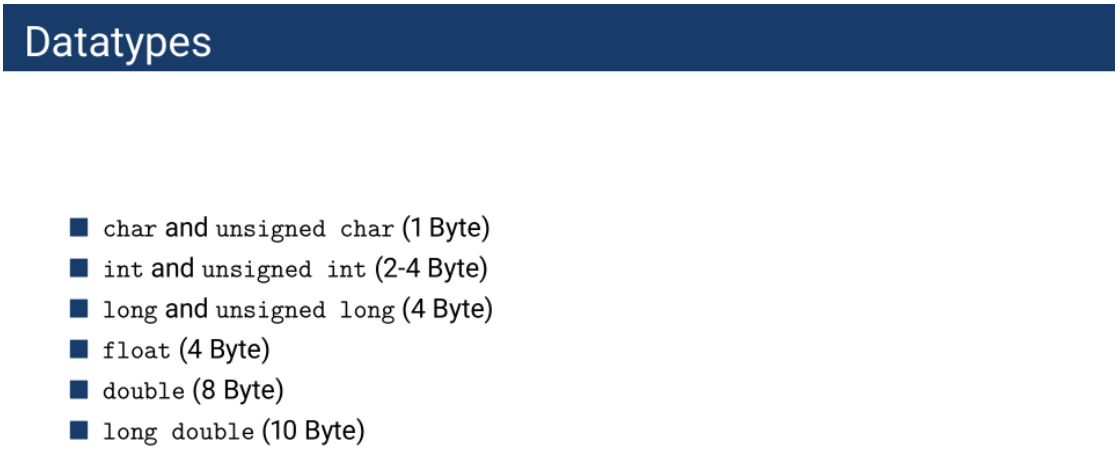
As I said, we assume you could solve this problem in the Java programming language, for example. Now, please think about what information you need from us so that you can write the program in the C programming language. Which of the typical programming language concepts are you missing? You are welcome to take some time now. Please pause the video, and if you like, you can also write down the function or program in Java first. Then please make a list of the topics you now need to write the program in C.

11 Agenda



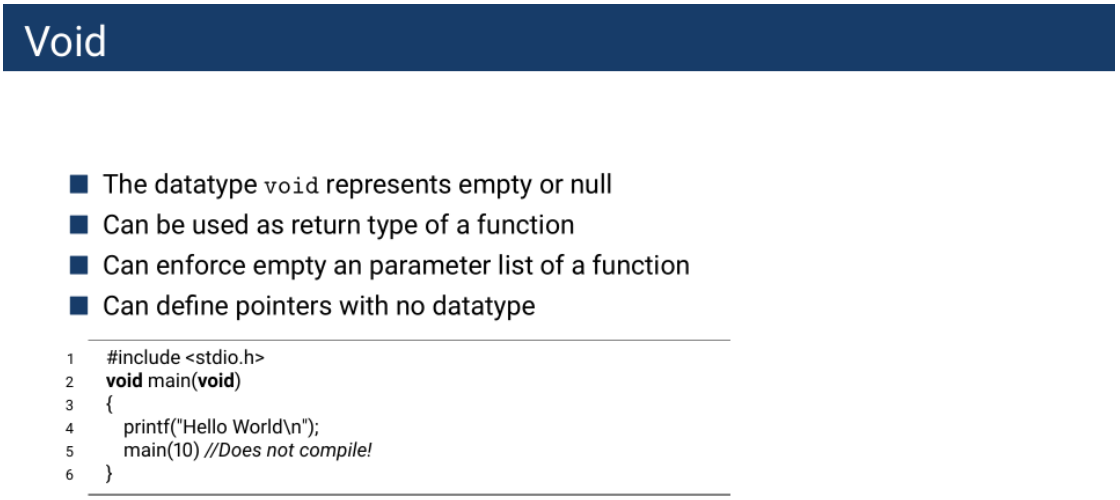
The following points should now be on your list of necessary topics. First, you need to know how to declare variables in C and what data types there are. Then, you need to know how to display the result of the algorithm on the screen. You also need to know the function to request data from the user. And finally, to solve this very simple task, you need the control structures, such as a conditional statement, because you need to be able to determine the two initial values. And you also need to know how to implement a loop. Of course, if your solution in Java already works with recursion, you would not need to understand loops in this first step. Finally, you would also need to know how to define a function with parameters. We will review all these points and then write the desired program for calculating the Fibonacci numbers together in C.

12 Datatypes



C is a statically typed programming language. This means that variables must be declared before their first use by specifying a data type, and the variable type does not change over the variable's lifetime. C knows the following data types; you can see the list next to me. The size in bytes is given in brackets, although this can vary slightly, especially with older computers or operating systems. The data type integer for whole numbers only takes up 2 bytes on some computers and four bytes on others. There are three data types for floating point numbers, which correspond to the corresponding precision lines according to IEEE Standard 754. As you can see, no separate data type for character strings exists. For example, other programming languages have something like the "string" data type. There is also no separate data type for truth values. In C, an integer value is typically used for truth values—the one stands for true and the 0 for false.

13 Void



Then, there is the void data type. This data type cannot be used for normal variables, but it can be used as a return type for functions and parameter lists. This data type stands for virtually nothing. So, if you declare a function with a void return type, it is not a function but rather a procedure. The caller of the function does not expect a result. The specification of void in the list of parameters expresses that the function does not expect any parameters. However, as in our first example, you could also express this by leaving the brackets empty. In a later unit, we will look at the concept of pointers. We will then see a way of using the Void data type for variables. The source code example beside me differs from our first example in line two. The function main should no longer have a return value; therefore, you cannot define an EXIT code within the function. In this case, the program always ends with the exit code null.

14 Printf

Printf

- `printf` prints something on the screen
- Placeholders can be defined inside the string
- Additional parameters will replace the placeholders
- Return value contains number of printed characters

```
1  #include <stdio.h>
2  int main(void)
3  {
4    int number = 10;
5    printf("Some number %d\n", number);
6    return 0;
7  }
```

Next, let's take a look at the Print F function. You can use this function to print something on the screen. If you want to use this function, include the Standard IO file as shown in line one. The Print F function can handle any number of parameters. The first parameter is always a text enclosed in double inverted commas. This text contains placeholders, which you can recognize by the percentage sign. Each placeholder is then replaced by a specific value for the screen output, and these values result from the other parameters when the function is called. After the percentage sign in the string is the letter D. This letter indicates that a decimal number is to be output here. The second parameter, "number," corresponds to a number. As usual, the backslash N stands for the line feed. We will need such a Print F instruction later to output the result of our calculation of the Fibonacci number on the screen. The return value of Print f is also a numerical value that indicates how many characters were output on the screen.

15 Printf Formatting - Specifier (1)

Printf Formatting - Specifier

% [Flag] [Width] [.Precision] [Prefix] Specifier

- `d` for decimal point (short/int/long)
- `o` for octal
- `x` for hexal
- `f` for floating point (float/double)

There are many setting options when specifying placeholders for data output with the Print F function. It is only necessary to define the so-called specifier. You can also use flags to explain how the output should be aligned and whether a sign should be output. You can specify the width of the output and the number of digits before and after the decimal point. All these specifications are optional. Only the specifier with which you specify the data type is mandatory. The Print F function cannot determine this using the parameters; you, as the developer, must specify this. The D here stands for decimal numbers, o for octal, i.e., base 8, x for hexadecimal numbers, i.e., base 16, and f for floating point numbers.

16 Printf Formatting - Specifier (2)

Printf Formatting - Specifier

% [Flag] [Width] [.Precision] [Prefix] Specifier

- s for char-sequence
- c for one char
- p for pointer adress
- % for percent

Then there is S for character strings, C for individual characters, and P for pointers. We will go into this in another unit. And if you want to output the percentage character, you have to write it twice.

17 Scanf

Scanf

- `scanf` reads something from default input stream
- Similar to `printf` format arguments are possible
- `scanf` can only read data, it can not write data
- Parameters define how to parse the input
- `scanf` requires addresses (`&`) of variables
- Return value contains the number of read characters

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int number = 10;
5     scanf("%d", &number); // Do not forget & before variable
6     return 0;
7  }
```

In our task, we requested that the user enter the number N for which we want to calculate the Fibonacci number. We, therefore, need a function with which we can accept user input. For this purpose, the Scan F function is used in C. The use of this function is very similar to the Print F function we have just learned about. You also specify a character string as the first parameter, delimited by inverted commas. As with the Print F function, you must define a data type within the character string to help the program write the input correctly in a variable. But now there is a big difference. Let's take a closer look at the example in line five. The percentage sign D defines the number we want to enter as a decimal number. The second parameter specifies the variable's name in which the input will be saved. But, and this is now very important, you must always enter the ampersand before the variable name. This does not pass the content of the variable number to the Scan F function but the address of this variable in the memory. The Scan f function cannot do anything with the value of the variable number but should save the user's input into a variable. To do this, the Scan F function must know where this variable is in the memory. This works via the address. Incidentally, the return value of the Scan F function contains the number of characters that were entered. By the way, you cannot use the Scan F function for a screen output at the same time, even if this seems possible at first glance.

18 Conditional Branch

Conditional Branch

- `if` is very similar to `if` in Java
- There is no datatype `boolean`, only integer values
- 0 ⇒ `false`
- Everything else ⇒ `true`

```
1  int number = 10;
2  if ( number == 10 )
3     printf("number is 10");
4  else if( number == 11 )
5     printf("number is 11");
6  else if( number )
7     printf("not zero");
```

Next, you need the control structure of the branch. When calculating the Fibonacci numbers, you must at least handle the two special cases for zero and one. So, if the parameter is zero, the return value of the Fibonacci function should be 0. Similarly, for parameter value 1, the branching in C is practically identical to the syntax in the Java programming language. The keyword is IF, and the condition is written in round brackets. Alternatives are started with Else, and you can combine Else and other IF statements, as seen in the example next to me. As soon as you no longer want to execute just one statement in a branch, you must first define a block with curly braces. A word of warning here. As C does not have a Boolean data type, any mathematical operation with an integer value is a valid expression for the condition—0 stands for false and any other value for true. So, if you implement an assignment instead of a comparison within the condition, for example, the result of this expression is the assigned number itself. The ELSE or the THEN branch is executed depending on whether the value is 0 or something else.

19 Loops

Loops

- Loops are also similar to Java
- `i` can be initialized inside the `for` loop since C99
- Be careful about the comma operator in C

```
1  int i;
2  for ( i=0; i<10; i++ ) {
3     printf("%d\n", zahl);
4  }
```

If you want to program the Fibonacci numbers iteratively rather than recursively, you now need the concept of loops. In C, there is both a for loop and a while loop. The for loop is practically identical to the for loop in Java. The loop variable is first given an initial value. After the first semicolon is the cancellation condition, and after the second is the instruction to change the loop variable. Inside the curly braces are the instructions that are to be executed repeatedly. If only one instruction is to be executed there, you can omit the braces. A note of warning at this point, too. The syntax may be correct if you only write a comma instead of a semicolon. In C, several loop variables are permitted.

20 Functions in C

Functions in C

- C supports other functions besides `main`
- Functions are comparable to static functions in Java
- Parameters take primitive datatypes as call-by-value
- A function has to declared before usage

```
1  void foo(int); // Prototyp
2  int main() {
3    foo();
4  }
5  void foo(int n) {
6    //something to do
7  }
```

Now, we come to functions in C. You can, of course, also write other functions in C apart from the Main function, and you can think of the functions as static methods in Java. We have already seen how to define the data type of the return value in the Main function. And here again, in line two, next to me. What you have yet to see is the definition of parameters. You can see an example of this in line five next to me. The Foo function expects a parameter of the integer data type. In the example next to me, you can see another special feature of the C programming language. A function can only be used once declared. Since the Foo function is called in the MAIN function, this Foo function is only defined later in the source code. We need the instruction in line one. We thus declare the Foo function as a so-called prototype. At this point, we are practically telling the compiler that there is a function Foo, but we are not yet saying exactly what this function Foo looks like. However, this declaration is enough for the compiler to translate this source text successfully.

21 Exercise 2: Fibonacci Numbers

Exercise 2: Fibonacci Numbers

- Write a program to calculate the fibonacci number `n`
- Take advantage of functions `scanf` and `printf`

So, now you have all the language elements of C together to implement the function for calculating the Fibonacci numbers yourself. Please take some time now to write a program in the C language using the knowledge you have just learned. This program must contain a function for calculating the Fibonacci numbers. The function expects a parameter and returns the result, i.e., the corresponding Fibonacci number. The main program should accept the input of the number N, then call the function to calculate the Fibonacci numbers, and finally display the result on the screen. Please press the button when you have finished or if you would like to view the result.

22 Screencast

- todo

So we start the editor to write this Fibonacci program. We first have to include standard I.O. again because we need some printf and scanf functions in our program. And let's start with the main function first. We print a command to the user that he has to type in the number n here. I'm sorry for the German text. We leave out the character return character at the end. Then we need a variable n that will take this parameter and this can be an unsigned in. The value of n is small so int is sufficient here. We know that n must be positive so unsigned is also okay. We call the function scanf here. We want to have a number and we have to pass the address of n. Then we declare a variable result that takes the Fibonacci number. We call the function and we print the result to the screen. First we type this number n again so the nth Fibonacci number and then we print the result. And here we must take care that it's a long value that we want to print to the screen. So the directive is ld not just d and we pass the two parameters here. The program is exited with a value 0. So next we have to write this function. And we accept this parameter n here. And first we take care of this like special cases. So if n is 0 then we return 0. If n is 1 then we return 1. And in all other cases we use a loop now. And to implement this using a loop we need some temporal variables here. So we declare a variable t1 and t2. And we initialize these two variables with 0 and 1. And we have a variable next that always takes the next element of this Fibonacci row. We use a for loop here. We start with the loop variable i equals 2. The other two cases were already handled in the if statements before. And we calculate the next value by the sum of the two previous values. And then we have to shift through the values. So the next t1 value equals t2 and the next t2 value equals next. And at the end we have then the result in the variable next. So let's try to compile this. OK here we have a problem in line 30. And there the curly brace is missing. And now we test the program first with 0 and then with number 1. And then

we can continue with 2, 3, 4 and 5. And this is all looking very good. So we can check the number 10 here. This is 55. And we can go to the Wikipedia page and just check if the results are correct here. We have the number n and the function value. So for 10 it's 55. So everything is correct. We can check for higher numbers. 21, this is also correct here. And then we can check for number 50. And here we have an overflow. But we take care of this later. OK next I would like to show you another implementation of this function using recursion. So we implement a second version. And this is much simpler of course. We only have to take care of the two start values n equals 0 and n equals 1. And in all other cases we just call the Fibonacci function for the two previous values. So for n minus 1 and for n minus 2. And we have to change the function call. And now I see there is a problem with the return type here. So this should be unsigned long. And also in the other function. Now this probably will also fix this problem with the overflow when n equals 50. So we start again to check now for some smaller values. And now we can call the function with recursion for value of 50. And here you can see one of the drawback of using the implementation with recursion. It is much slower because we have a lot of function calls here. So in each function call we call this recursion function twice. So this takes a minute or so to calculate the result. And we go back to the version that uses loops. And I just want to show you the difference in the performance here. So we change this back to the first implementation. We have to compile this and we start the program with 50. And you see we have the result immediately.

23 Summary

- C is an imperative programming language
- Data types, control structures, and functions
- Syntax is similar to Java (or vice-versa)
- Memory management is more difficult (another unit)

This was the first step in our introduction to the C programming language. Today, you have learned that C is an imperative programming language that allows you to program very close to the computer's hardware. Software written in the C programming language is typically extremely fast. We have introduced them to the programming language basics in terms of data types, control structures, and procedures for declaring and defining functions. You have seen that the syntax for the language constructs presented today is very similar to the syntax of Java. It is the other way around; Java was a much younger programming language. Java's syntax was strongly orientated towards C. In this first introduction, we have already pointed out several times that programming in C can be very dangerous under certain circumstances because you can write directly into the memory. Memory management is extremely complex, and we will discuss this in detail later.

24 Literature

Literature

- Brian W. Kernighan, Dennis M. Ritchie: The C Programming Language. Prentice Hall, 1988.
- Avelino J. Gonzalez: Computer Programming in C for Beginners. Springer, 2020.
- Helmut Schellong: Moderne C-Programmierung. 2. Auflage, Springer Vieweg, 2013.
- Jürgen Wolf: C von A bis Z. 4. aktualisierte und überarbeitete Auflage, Rheinwerk Computing, 2020. (Die 3. Auflage ist als Open Book verfügbar.)