

Data Structures and Algorithms

Course 1 Slides

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

Linus Torvalds

Administrative Info

Lecturer: Radu Hobincu (radu.hobincu@upb.ro), Senior Software Engineer at Ixia

Lab Assistant: Gabriel Neagoe (neagoegab@gmail.com), Senior Software Engineer at Infineon Technologies

Course: 14 lectures covering C11 syntax, data structures and algorithms

Laboratory: 7 meetings designed to improve programming skill and algorithms understanding and implementation

Resources:

- Development virtual machine (Linux Mint) for VirtualBox - LinuxDev: <ftp://zeus.arh.pub.ro/pub/VirtualMachines>
- Extensive Algorithms Course- Cormen: <https://drive.google.com/file/d/oBoKC3c5boTpYLXY4UE9qUWJaTHM/edit?usp=sharing>

Administrative Info (cont'd)

Grading System (from a total of 100 points):

- 60 points are awarded for 6 home assignments each weighting 10% of the final grade, specified during the lecture in every even week, starting with week 2
 - Homework assignments are electronically submitted and they are compared with each other with a anti-plagiarism tool; if two or more copies are detected, all submissions in question will be discarded and no points will be awarded.
 - If a student accumulates more than 50 points from homework assignment, the lecturer may offer a final grade to the student with the option of him not attending the final exam.
- 20 points are awarded by the lab assistant for activity during the lab
- 20 points are awarded during the final exam from which a **minimum of 10 points** must be earned in order to pass the subject
- **50 or more points** are required in order to pass this subject

Course Contents

Course 1: High Level Programming Languages: Compilers and Integrated Developing Environments; C Language review, existing standards - C11; C vs. C++;

Course 2: C Syntax Review – variables, constructs, functions and pointers; Memory allocation: heap, stack and global variables

Course 3: Pseudo-code; Algorithms and complexity; Quantifying algorithm quality; Recursion – Divide et Impera; Hanoi towers problem

Course 4: The array: advantages and disadvantages; Algorithms on the array: Sorting (Selection Sort, Bubble Sort, Merge Sort, Quick Sort) and searching (Binary search)

Course 5: The stack and the queue; Reversing a word

Course 6: Lists: advantages and disadvantages; Insertion, deletion, searching, iteration and memory allocation

Course 7: Hash Maps: advantages and disadvantages; Hash functions and hash collisions; Insertion, deletion, searching, iteration and memory allocation

Course Contents (cont'd)

Course 8: Trees and binary trees; Tree sort; Tree depth-first and breadth-first search

Course 9: Graphs: directed and undirected; Storing graphs; Connected components

Course 10: The minimum spanning tree of an undirected graph; Minimal cost route; Eulerian and Hamiltonian graphs;

Course 11: Backtracking – generate all permutations of a sequence

Course 12: Dynamic programming

Course 13: Dynamic programming

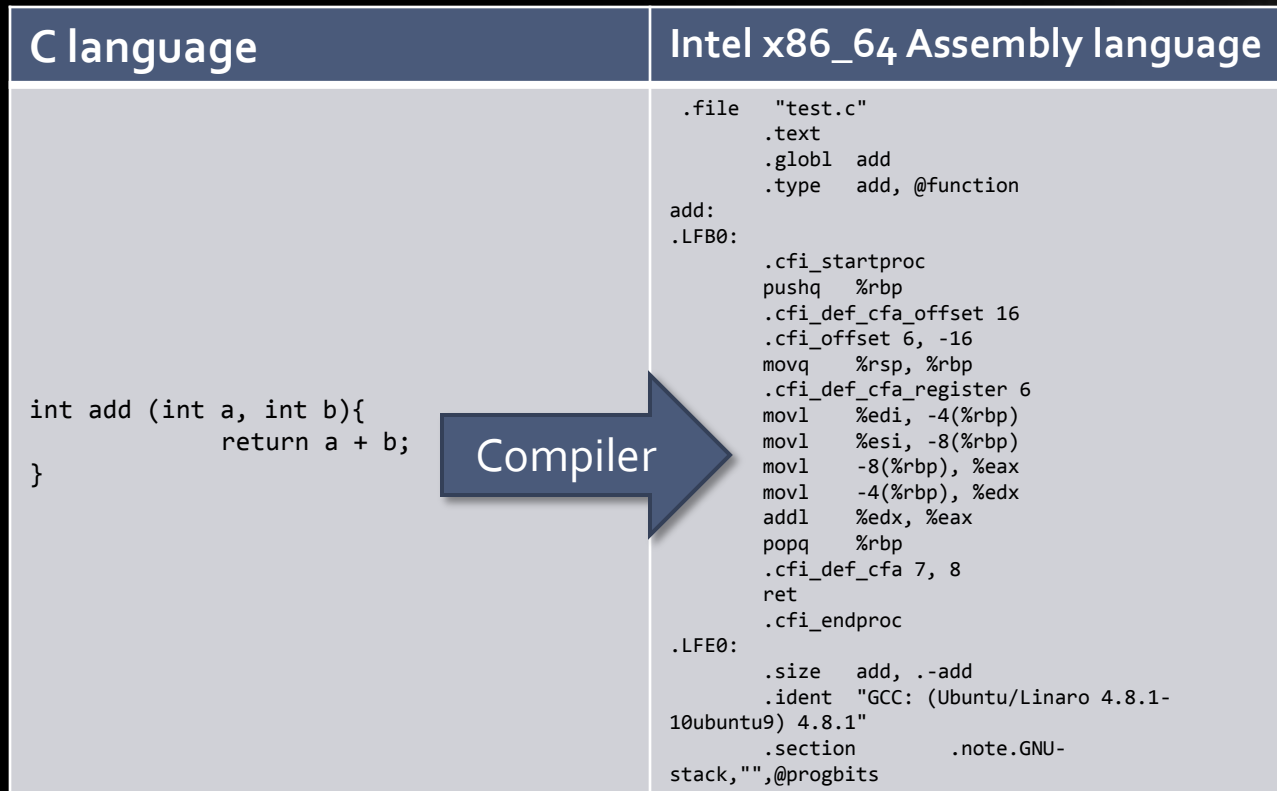
Course 14: Q&A

High Level Programming Languages

- Digital processors have a very simple set of instructions they can understand so **developing a complex algorithm in assembly requires a very large number of steps**
- Each processor understands its own set of instructions but not others' so **developing a software in assembly that runs on any processor is impossible**
- High level languages are used to **accelerate software development** by providing a **logical, compact and unified syntax** for describing processor behavior
- There are hundreds of high level programming languages but usually each of them is optimized for a certain application domain:
 - C – system/ kernel programming
 - C++ – complex applications that require speed
 - SQL – database design and management
 - Java – complex applications that are not speed sensitive but are very dynamic and require high portability, etc.

Compilers

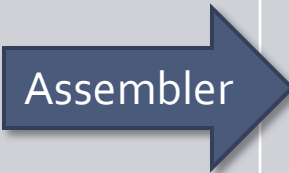
- Compilers are software programs that **translate high level languages to assembly languages**, so they are, in fact, automated translation tools between two formal languages
- Compilers have a **front-end**, that depends on the HLL, and a **back-end**, that depends on the processor/ set of instructions they are compiling for



Assemblers

- Assemblers are software tools that **encode the text mnemonics of an assembly language instruction sequence into machine code**
- Assemblers usually generate .obj files in Windows and .o files in Linux
- Assemblers are **CPU specific**

Intel x86_64 Assembly language	X86_64 machine code
<pre>.file "test.c" .text .globl add .type add, @function add: .LFB0: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 movl %edi, -4(%rbp) movl %esi, -8(%rbp) movl -8(%rbp), %eax movl -4(%rbp), %edx addl %edx, %eax popq %rbp .cfi_def_cfa 7, 8 ret .cfi_endproc .LFE0: .size add, .-add .ident "GCC: (Ubuntu/Linaro 4.8.1- 10ubuntu9) 4.8.1" .section .note.GNU-stack,"",@progbits</pre>	<pre>0000000000000000 <add>: 0: 55 1: 48 89 e5 4: 89 7d fc 7: 89 75 f8 a: 8b 45 f8 d: 8b 55 fc 10: 01 d0 12: 5d 13: c3</pre>

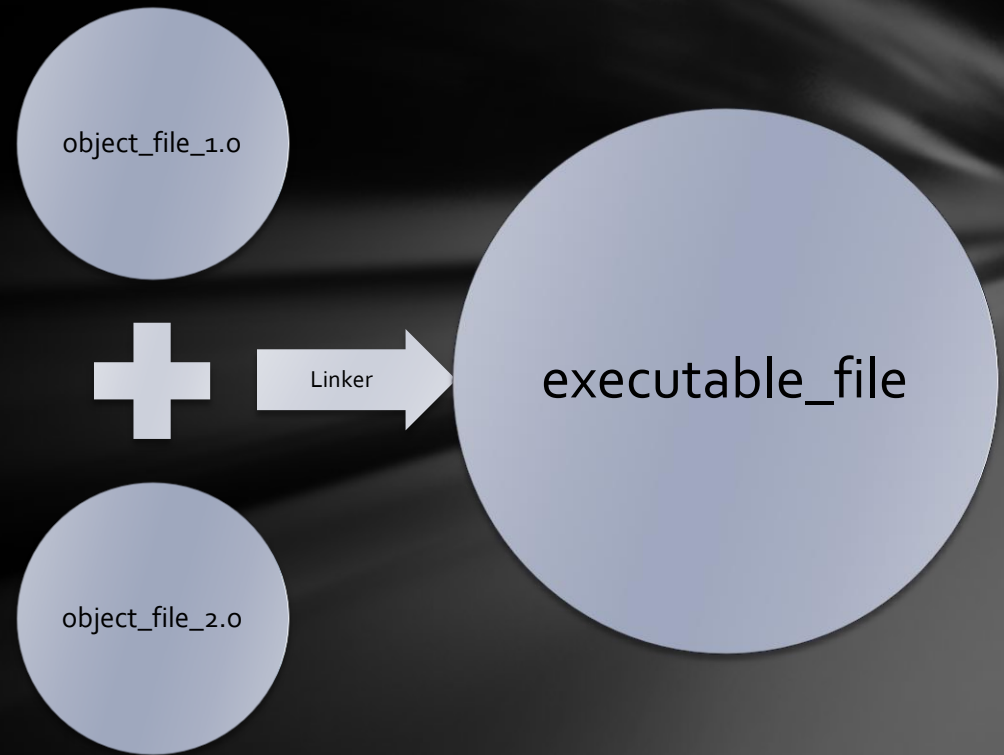


Linkers

Linkers are used to “glue” one or more object files together and generate an executable file.

Linkers are required because not all the functions that you need in your program are compiled in the same file.

E.g.: the printf function is called to print on the screen, but it is defined in stdio.h header and compiled in the standard C library.

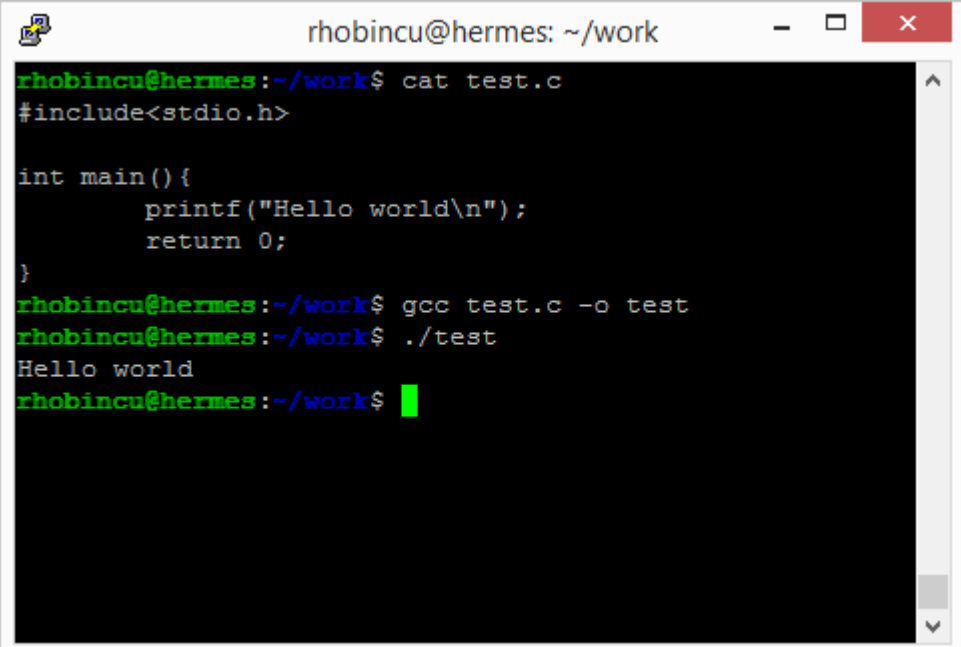


GCC – The world's most used C compiler

GCC (GNU Compiler Collection) is the most famous and most used C compiler in the world. It is open-source and it is free to use.

It supports not only C, but also C++, Objective-C, Fortran, Ada, Java and Go.

*GCC is a **command line tool!***

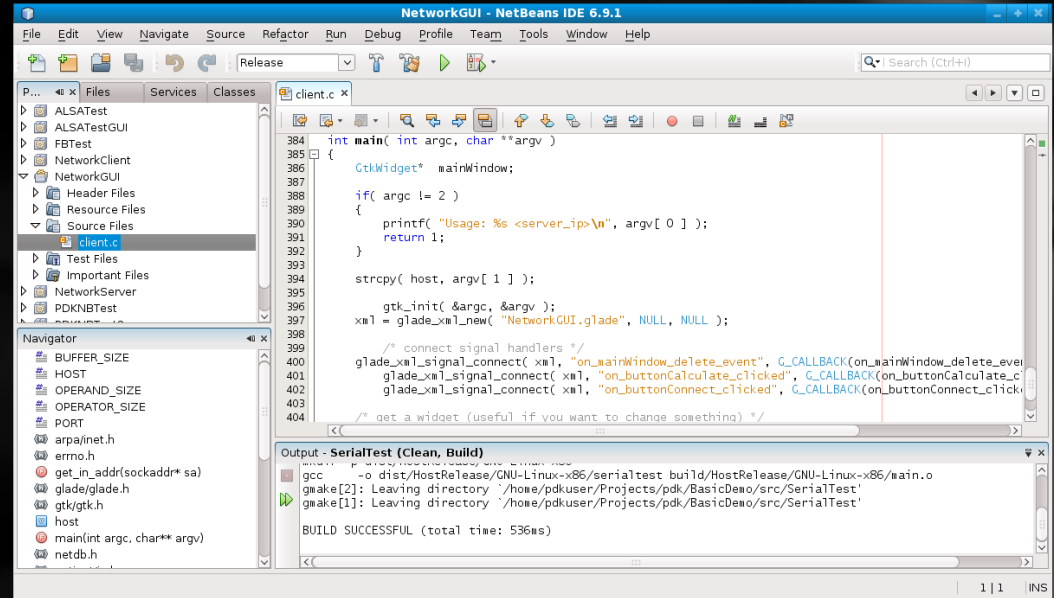


```
rhobincu@hermes: ~/work
rhobincu@hermes:~/work$ cat test.c
#include<stdio.h>

int main(){
    printf("Hello world\n");
    return 0;
}
rhobincu@hermes:~/work$ gcc test.c -o test
rhobincu@hermes:~/work$ ./test
Hello world
rhobincu@hermes:~/work$
```

Integrated Development Environments (IDEs)

IDEs are graphical user interfaces that make use of a third party compiler to enable a programmer to write the code and compile within the same application, instead of using an external text editor and compiling using command line.



C Language Review

- The initial development of C occurred at AT&T Bell Labs between 1969 and 1973
- In 1989, the C standard was ratified as ANSI X3.159-1989 "Programming Language C". This version of the language is often referred to as ANSI C, Standard C, or sometimes C89.
- The C standard was further revised in the late 1990s, leading to the publication of ISO/IEC 9899:1999 in 1999, which is commonly referred to as "C99"
- In 2007, work began on another revision of the C standard, informally called "C1X" until its official publication on 2011-12-08. The **C11 standard** adds numerous new features to C and the library, including type generic macros, anonymous structures, improved Unicode support, atomic operations, multi-threading, and bounds-checked functions.
- The reference for the C11 standard can be found here: <http://en.cppreference.com/w/c> and standard itself here: <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf>

What is C++?

- C++ is a different language that introduces specific new features:
 - **Object oriented constructs:** classes, inheritance, virtual functions, overriding
 - Function overloading
 - Operator overloading
 - **Streams**
 - Templates
 - Exception handling
- The most important feature introduced in C++, are the availability of the object oriented constructs; if you're not using classes, then you should not use C++ or claim you know it

Thank you!

* for next time, review C language syntax: variables,
functions, constructs and operators