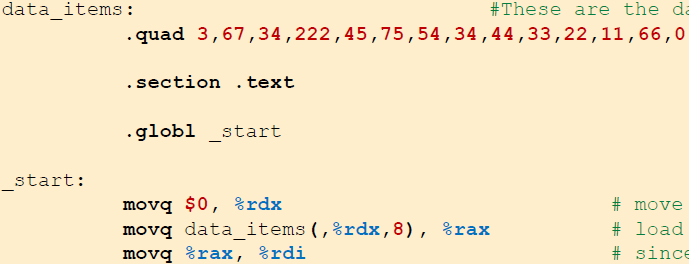
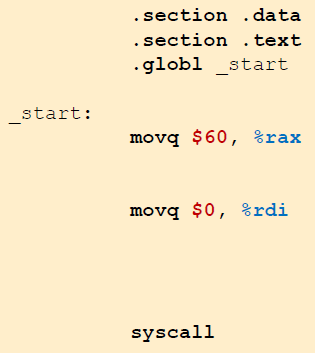
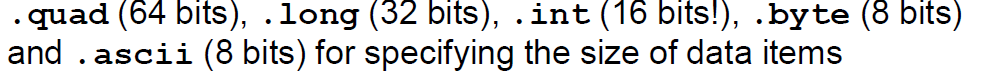
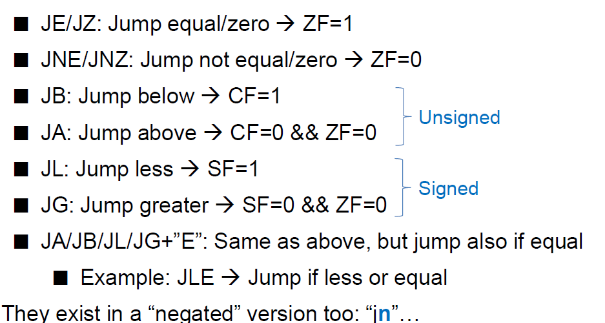
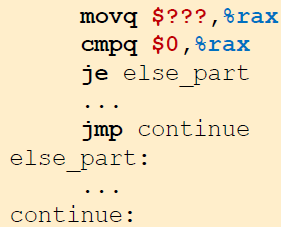
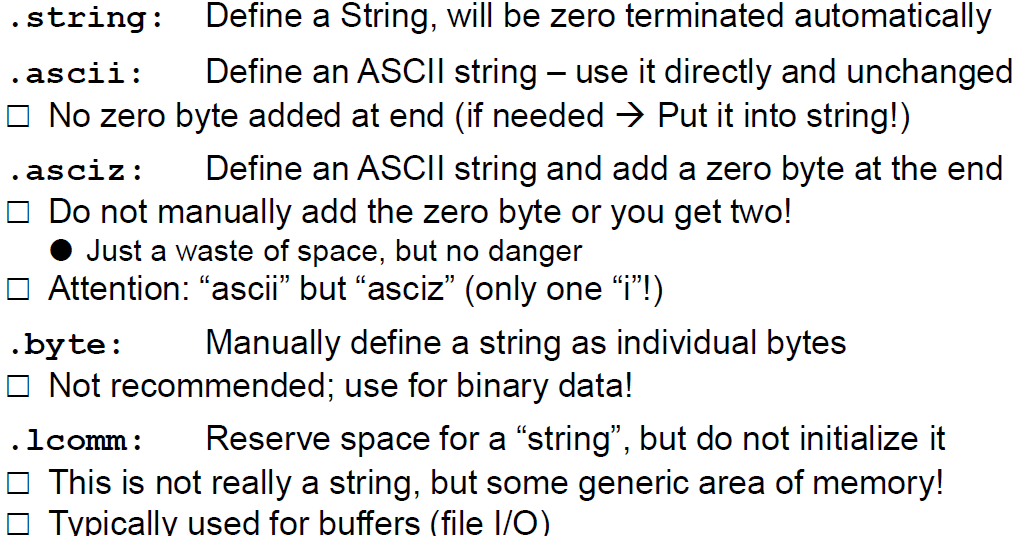
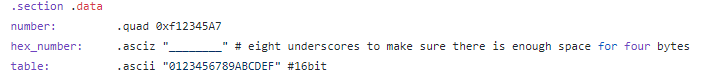
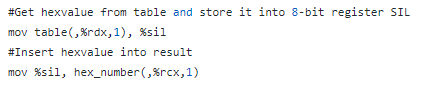
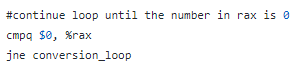
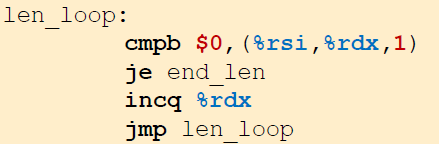
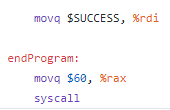
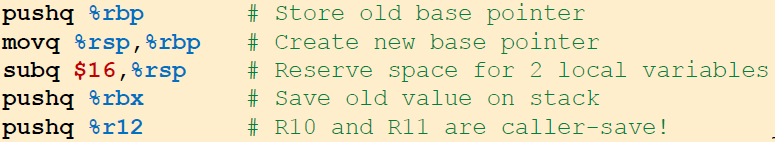
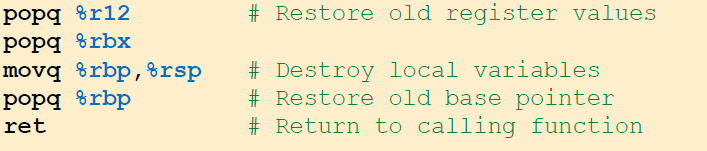
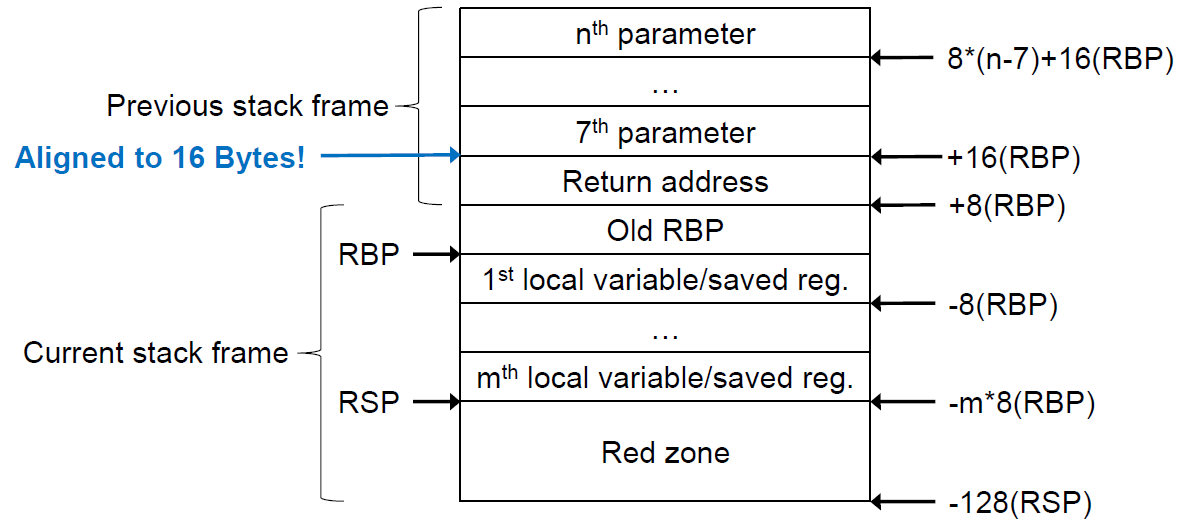
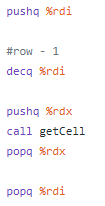
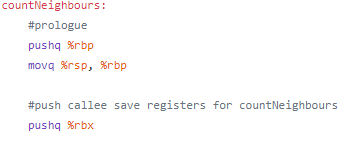
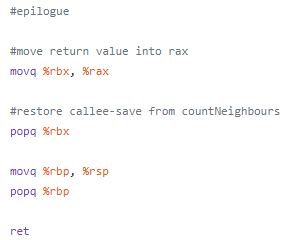
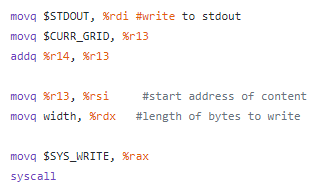
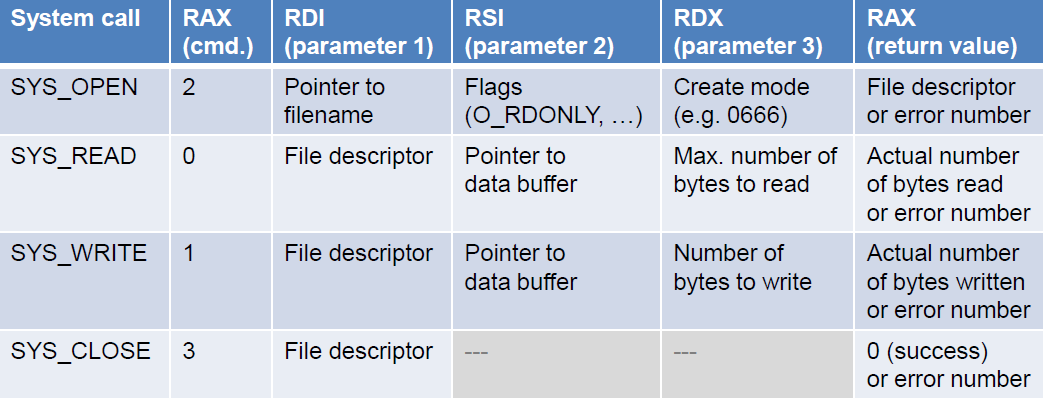
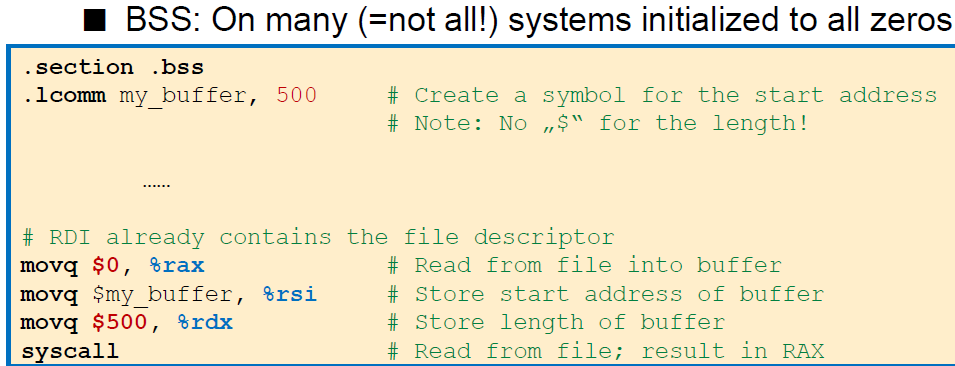
**Systems-Programming**

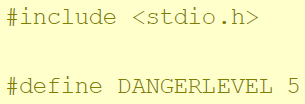
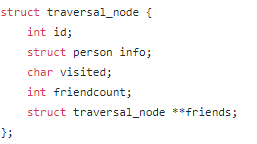
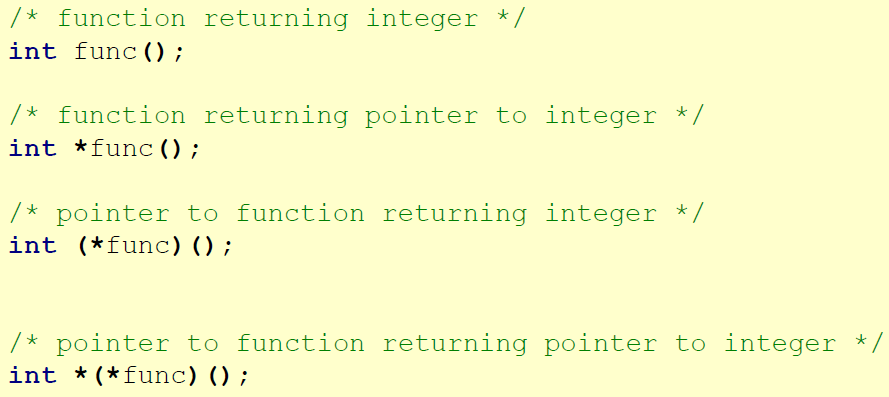
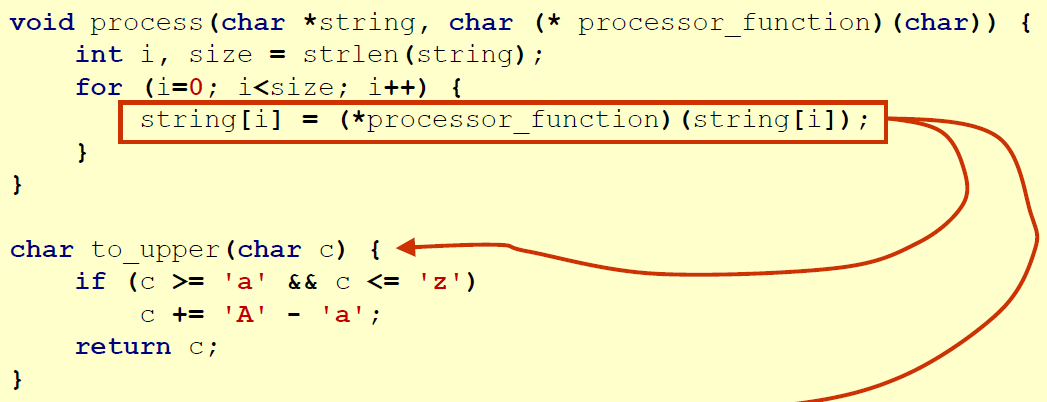
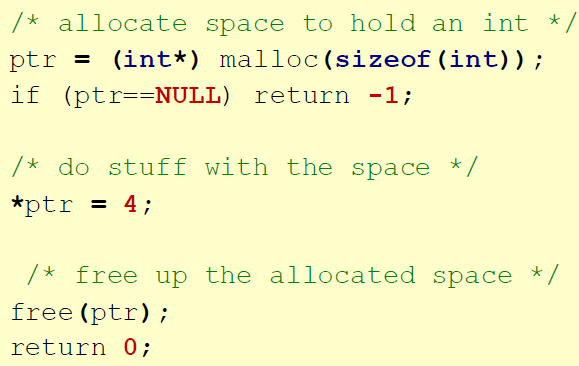
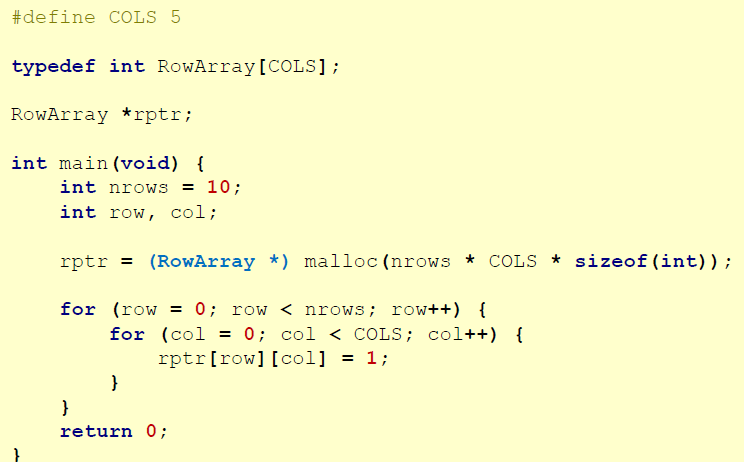
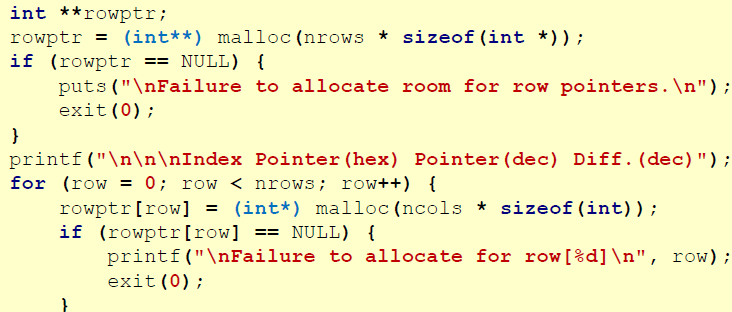
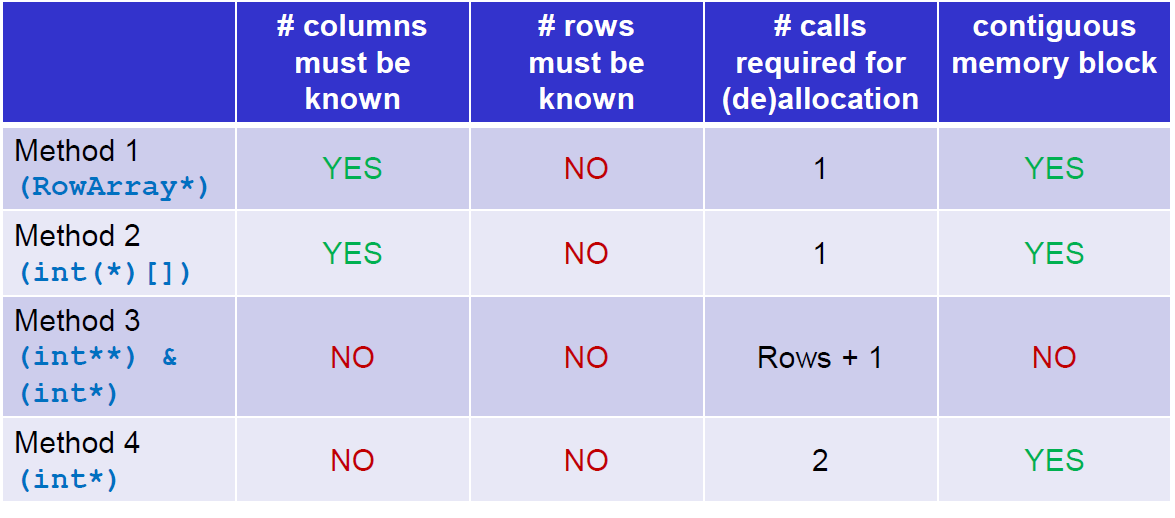
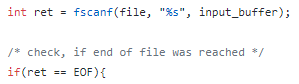
**Assembler**

* Word = 8 bytes on 64-bit CPU
* Registers
  + General Purpose (RAX, RBX, RCX, …)
  + Special Purpose (RBP, RSP, RIP)

|  |  |
| --- | --- |
|  |  |

* Data Access Methods
  + Immediate
    - 
  + Register
    - 
  + Direct
    - 
    - Value at address 12 in RAM
  + Indirect
    - 
    - Value at address inside rbx
  + Indirect with base value
    - 
    - Value add address 4+<value of rbx>
  + Indirect Indexed
    - 
    - Value at data\_items + offset (0) + rdi \* 8 )
    - 
* Program structure
  + Sections:
    - .data = Program constants
    - .bss = Buffers and dynamically allocated things
    - .text = Code
  + Start
    - .globl <label> defines the entry point of the program
  + Defining global variables
    - 
  + Conditional jumps (If/While statements)
    - 
  + Defining text
    - 
  + Functions
    -  to show that the code beneath specific label is a function -> followed by label for function
* Examples
  + Define variables 
  + Get value at specific position and write to specific position
  + Check loop condition 
  + Calculate string length
    - 
  + Quit program
    -  rdi = return value
* Function prologue/epilogue
  + Must be done first in a function (in our case everytime)
  + 
    - Save all callee-save registers!!!
  + Must be done last in a function
  + 
    - Restore all callee-save registers!
* Stack
  + Storage of return addresses, parameters, local variables etc.
  + Stack is at the top addresses of memory (each process/thread has a separate stack)
    - Grows from high towards low addresses
    - Has a size limit
    - RSP points to top of stack
      * Pushq decrements RSP
      * Popq increments RSP
* Calling convention (SystemV AMD64 ABI)
  + Function input
    - First six parameters 
    - Rest are pushed on the stack in reverse order
  + Function output
    - Return value in RAX (+ RDX if 64-bit are not enough)
  + Caller-save registers (saved before calling the function by the caller)
    - 
    - Edit: R8-R11 instead of R1-R11!!!
  + Callee-save registers (must be saved in function prologue and restored in epilogue if used)
    - 
  + Stack frame of a function call
* Examples
  + Caller-save registers
    - 
  + Callee-save registers and prologue
    - 
  + Callee-save registers and epilogue
    - 
* Files
  + Open and write to file (in this case, the file is STDOUT)
    - 
    - 
    - Parameters of different FILE Syscalls
* Buffers
  + 

**C**

* Preprocessor
  + 
* Implicit conversions 
* Arrays
  + No length attribute like in Java
  + Define size with preprocessor constants or dynamic allocation
* Structs
  + 
  + Using pointer: -> for accessing elements
  + Using value: . for accessing elements
* Typedefs
  + Give type an additional name
  + 
* Pointers
  + \*ptr – dereference pointer
    - 
  + &ptr – get reference of pointer
    - 
  + Pointer to function
    - 
  + Call function as parameter in function
    - 
* Dynamic allocation
  + Always check, if returned pointer is NULL
  + Type of malloc/calloc/… is (void \*)
    - Implicit type conversion done automatically
    - Explicit type conversion can be done but be are of wanted type!
  + malloc
    - 
  + calloc
    - 
    - 1 = amount of elements: n\*sizeof(<whatever>);
    - Typecast can be left out for implicit conversion
  + realloc
    - 
  + Free
    - 
  + Methods for dynamically allocating memory
    - Pointer to one-dimensional array
    - Number of elements not known at compile time -> pointer to pointer (two-dimensional pointer) 
      * Free() call for every entry in the array needed and for rowPtr as well
    - 
* Files
  + Type FILE
  + Always check, if NULL value occurred after an operation
  + Open
    - 
    - Permissions
      * r … open existing file for read-only
      * w … open existing file for write-only
      * a … append (if file not exists -> create)
      * r+ … read and write
      * w+ … read and write -> if file exists, it is truncated to 0 length, if not -> new file created
      * a+ … read and append – if not exists, new file is created
  + Close
    - 
  + Reading line by line
    - In a loop:
    - 
    - EOF represents, that End-of-file was reached
    - Input\_buffer is of type char[BUFFER\_LENGTH]
    - Split values with strtok
      * 