CHALMERS
UNIVERSITY OF TECHNOLOGY

Machine-Oriented Programming

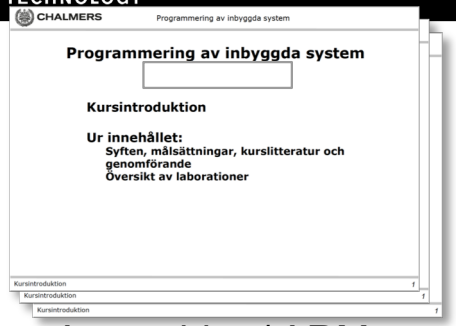
Introduction to C-Programming

Pedro Trancoso
ppedro@chalmers.se

Original slides by Ulf Assarsson

Content

- Introduction: Environment, literature, etc.
- Different languages
- Process: from source to executable
- Types1: char, short, int, unsigned
- Declaration, assignment, type conversion (casting)
- Operators, bitwise, expression evaluation
- Program flow: conditional (if), loops (for, while)
- Functions, function call, parameters, return values



Assembler / ARM-lectures

5 C-lectures

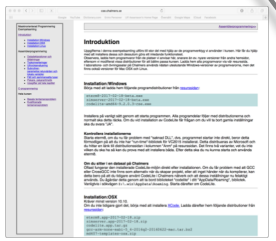
Examples
C – Exercise tasks.
(online)

Same webpage

Textbook

Workbook

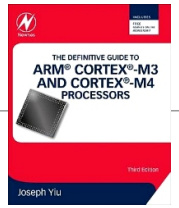
LabPM
(online). 5 labs.



Examples
(exercises online).

If you like:

ARM:



The Definitive Guide to ARM® Cortex® -M3 and Cortex-M4 Processors, Third Edition, Joseph Yiu, ARM Ltd, Cambridge, UK.

C:

- The C programming Language (ANSI-C)
  - https://hassanolity.files.wordpress.com/2013/11/the_c_programming_language_2.pdf
- Vägen till C, Skansholm
- C från början, Skansholm

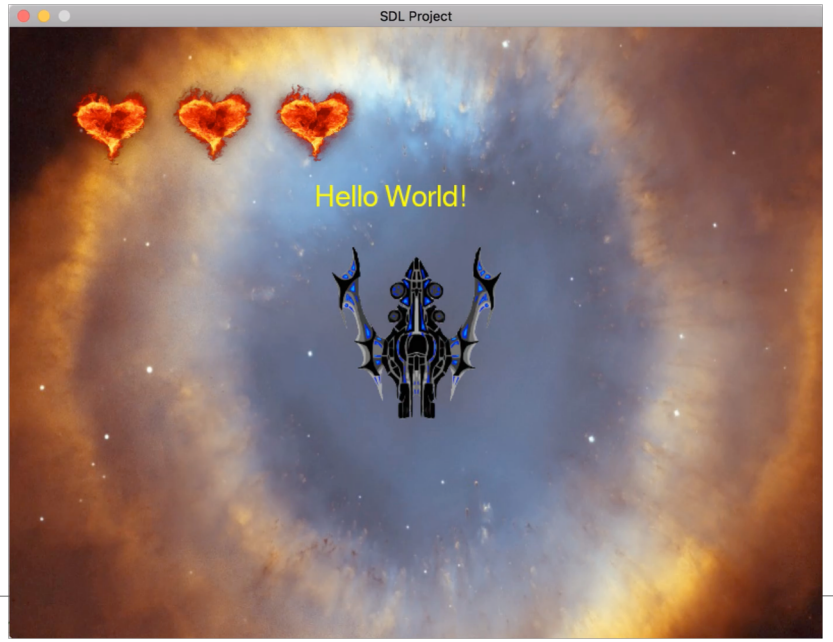
Textbooks?

- The workbook is equivalent to the textbook.
  - Labs – learn how to run things for the target hardware and simulator.
  - Lectures – to understand the workbook (+ lab).
  - Online Example collection:
    - Exercises in assembler/C
- C:
  If you need a book for C:
  - The C programming Language (ANSI-C)
    - https://hassanolity.files.wordpress.com/2013/11/the_c_programming_language_2.pdf
  - Vägen till C, Skansholm, 2011.
  - C från början, Skansholm, 2016.
  - C reference card ANSI (on the course's website under "resurssidan")
    - http://www.cse.chalmers.se/edu/resources/pinsys/ebook/c-refcard.pdf

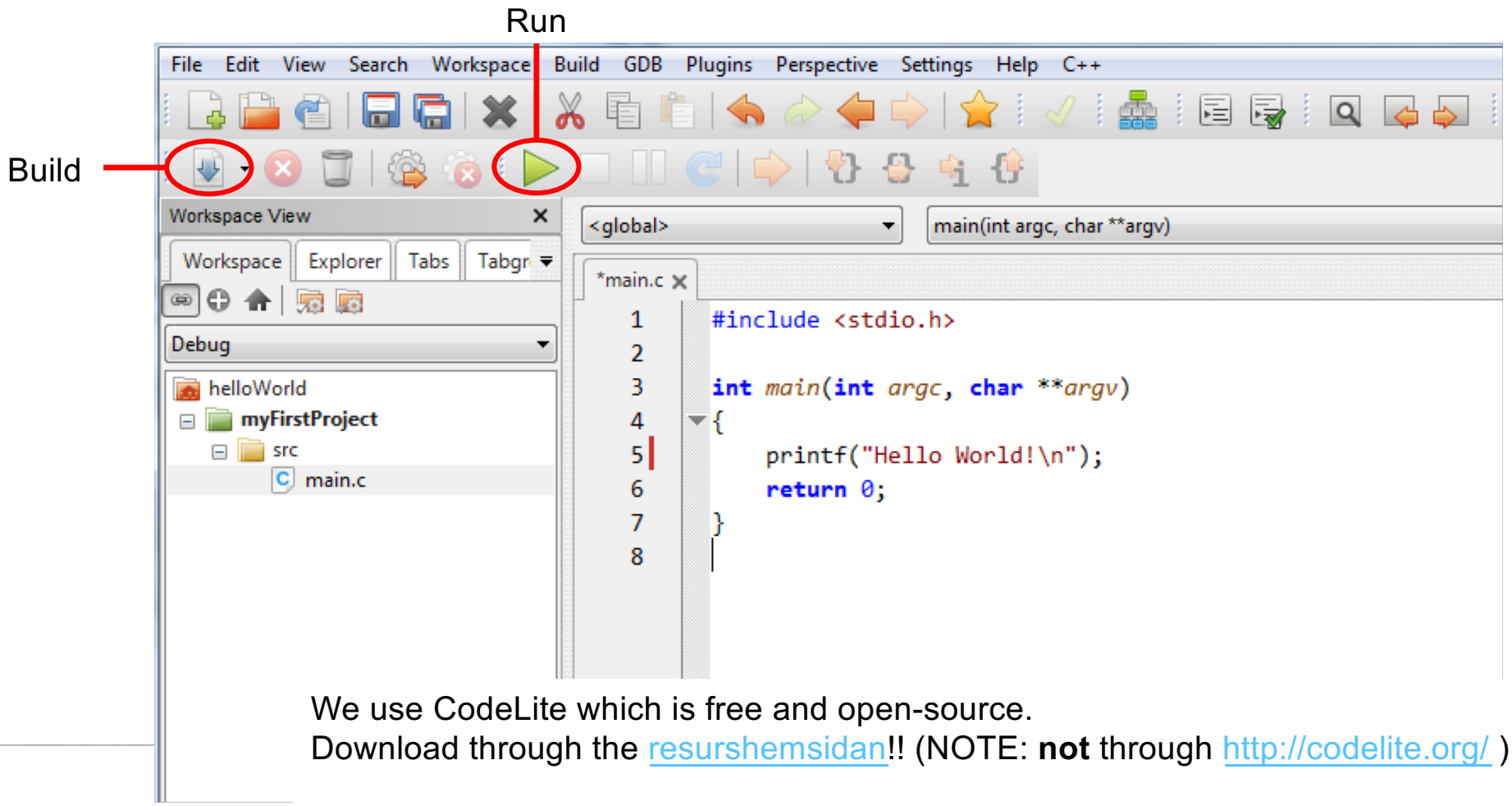
C – Exercises

- Online – see the link on the course homepage for today's C lectures.

  (The first week's exercise also contains links to alternative beginners C exercises online.)

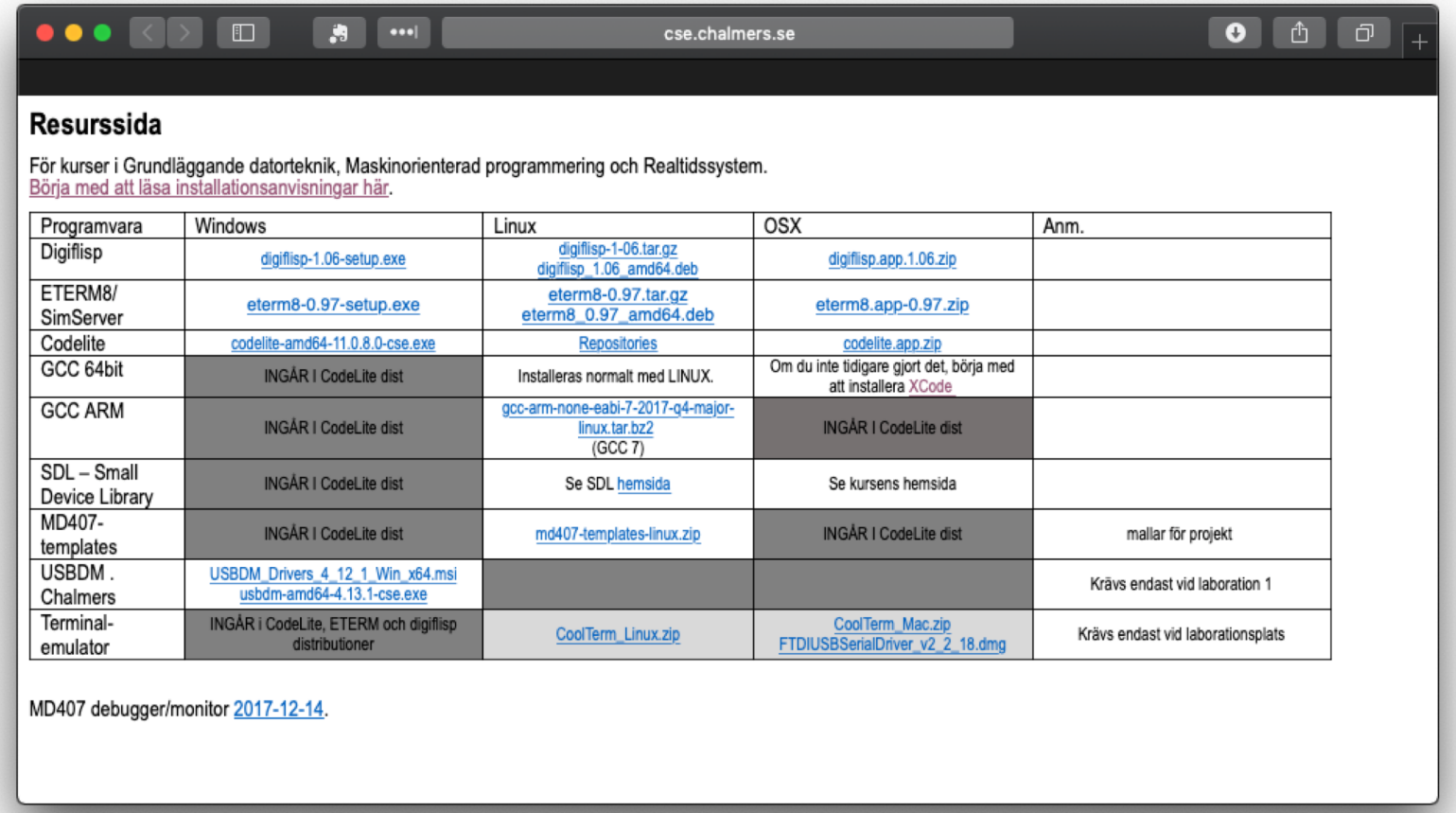


Integrated Development Environment (IDE)



We use CodeLite which is free and open-source.
Download through the resurshemsidan!! (NOTE: **not** through http://codelite.org/)

CodeLite – Download through Home->Kursmaterial->Programvaror



Resurssida

För kurser i Grundläggande datorteknik, Maskinorienterad programmering och Realtidssystem.
Börja med att läsa installationsanvisningar här.

Programvara	Windows	Linux	OSX	Anm.
Digiflisp	digiflisp-1.06-setup.exe	digiflisp-1-06.tar.gz digiflisp_1.06_amd64.deb	digiflisp.app.1.06.zip	
ETERM8/ SimServer	eterm8-0.97-setup.exe	eterm8-0.97.tar.gz eterm8_0.97_amd64.deb	eterm8.app-0.97.zip	
Codelite	codelite-amd64-11.0.8.0-cse.exe	Repositories	codelite.app.zip	
GCC 64bit	INGÅR I CodeLite dist	Installeras normalt med LINUX.	Om du inte tidigare gjort det, börja med att installera XCode	
GCC ARM	INGÅR I CodeLite dist	gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2 (GCC 7)	INGÅR I CodeLite dist	
SDL – Small Device Library	INGÅR I CodeLite dist	Se SDL hemsida	Se kursens hemsida	
MD407-templates	INGÅR I CodeLite dist	md407-templates-linux.zip	INGÅR I CodeLite dist	mallar för projekt
USBDM . Chalmers	USBDM_Drivers_4_12_1_Win_x64.msi usbdm-amd64-4.13.1-cse.exe			Krävs endast vid laboration 1
Terminal-emulator	INGÅR i CodeLite, ETERM och digiflisp distributioner	CoolTerm_Linux.zip	CoolTerm_Mac.zip FTDIUSBSerialDriver_v2_2_18.dmg	Krävs endast vid laborationsplats

MD407 debugger/monitor 2017-12-14.

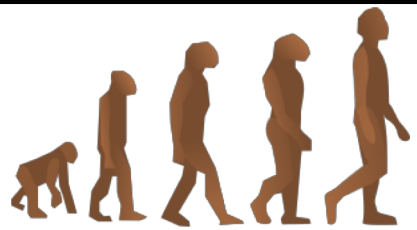
From the terminal

```
> gcc -o hello.exe main.c
> hello.exe
Hello World!
>
```


Build


Run

Programming Language Evolution



www.openclipart.org

- 1949: Short Code, 1:st high level language
- 1950s: Autocode, early 50'ies
- 1957: Fortran, IBM
- 1958: Lisp
- 1960: Cobol 60 (Common Business-oriented language)
- 1964: BASIC
- 1960: ALGOL 60 (ALGOrithmic Language 1960)
- 1960s: Simula
- 1969: C
- 1972: Prolog
- 1975: Ada
- 1975: Pascal
- 1978: ML

C was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Labs, and used to re-implement the Unix operating system

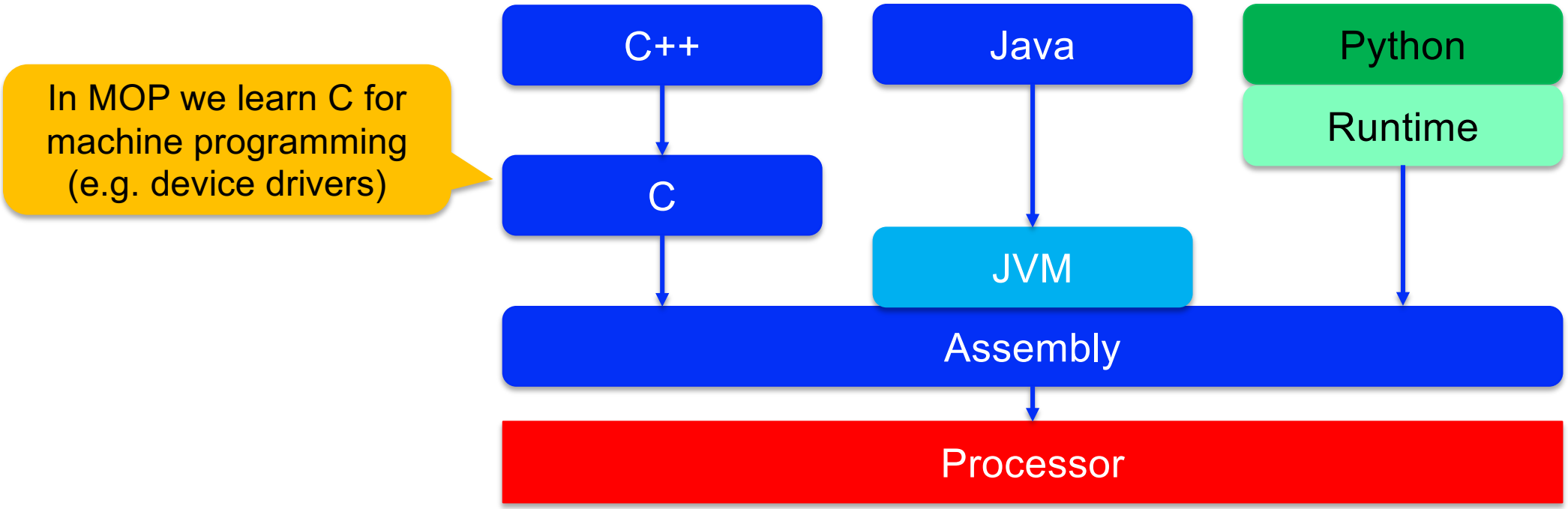
```c
#include <stdio.h>

int main()
{
   printf("Hello World!\n");
   return 0;
}
```

Top 3 programming languages you used?

Vote View

Different languages

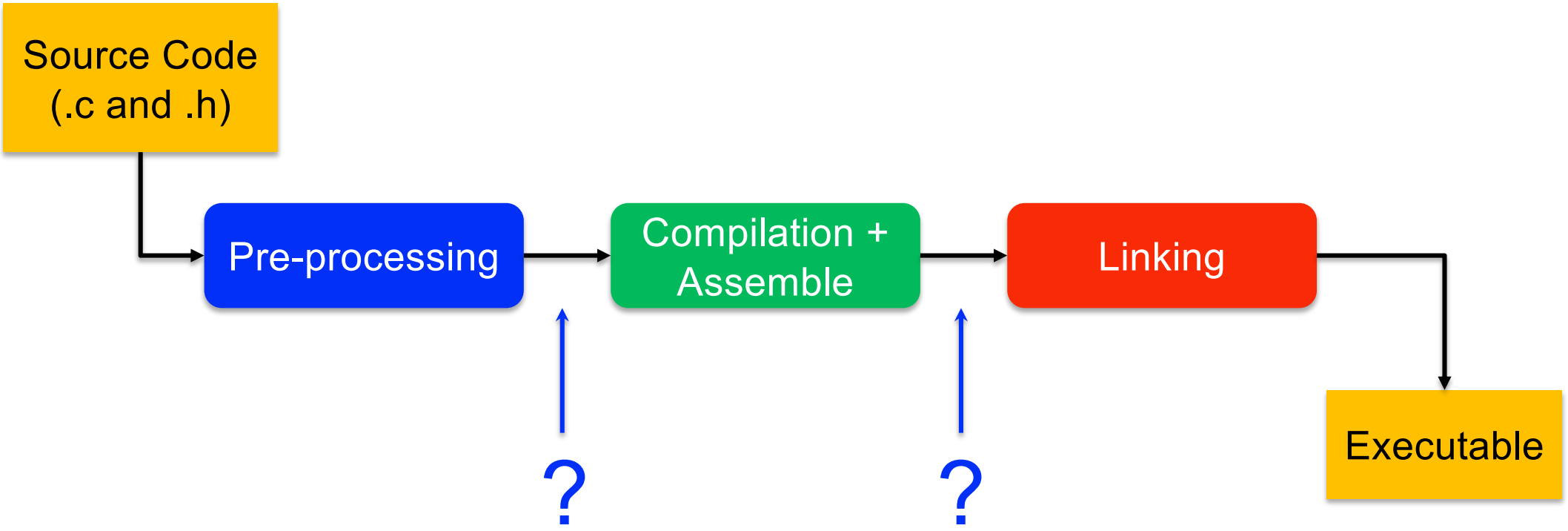


C vs. Java

```
struct Course {
    char* name;
    float credits;
    int numberOfParticipants;
};
```

1. C: Does not have classes.
   Has struct for composite data type.
2. Booleans are NOT a type. There is no true/false. 0 is false and a value != 0 is true.
   [you usually define TRUE/FALSE as 1 and 0. Even so, 1 && 1 is not necessarily 1! “implementation-specific for the compiler”]
3. No array boundary checks! 😈-vs-😇
4. No exception handling – try/catch
5. No polymorphism or overloading
6. Operations with operands of different types (and type conversion!)
7. Compilation! 😈-vs-😇

From source code to executable



Pre-Processor

```
// main.c
#include <stdio.h>
#include "foo.h"

#define MAX_SCORE 100
#define SQUARE(x)  (x)*(x)

int main()
{
   printf(”Highest possible points are %d\n", MAX_SCORE);
   printf(”Square of 3 is %d\n", SQUARE(1+2));
   printf("x is %d", foo(0));
   return 0;
}
```

Copy-paste from header files

String find-and-replace

The pre-processor works on the source code at the text-level

Note: If you want to check the output of the pre-processor phase run gcc –E!

Compiling

- Processes one .c-file at a time:
  - Creates an object file (.o-file) (e.g. gcc -c), per .c-file, containing:
    - Machine code instructions
    - Symbols for addresses
      - For functions/variables in the object file.
      - For functions/variables in another object file/library.

Note: .o file is in binary format so you you need to generate the assembly explicitly to see the instructions and labels generated (e.g. gcc –S)

main.s:

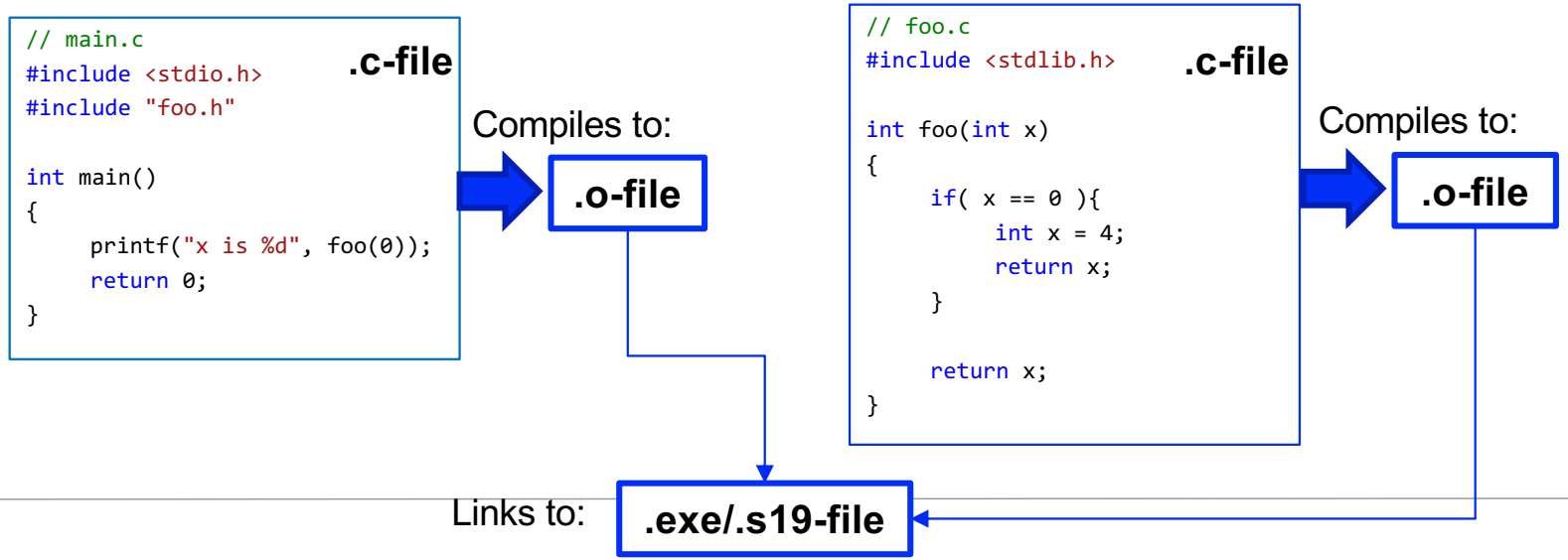
```
myFkn:      mov   r3, #0
            …
            bx    lr

main:  bl     myFkn
            bx    lr
```

What is the difference between ***static*** and ***dynamic*** linking?

Linking

- Merge multiple object files into one executable file (.exe/.s19)
- Translate the symbols to (relative) adresses



Types & declarations

```
char c;

short s;

int i, j;

long x;

x = 2;
c = 'a';
```

```
uns

uns

uns

unsigned long x;

x = 2;
c = 'a';
```

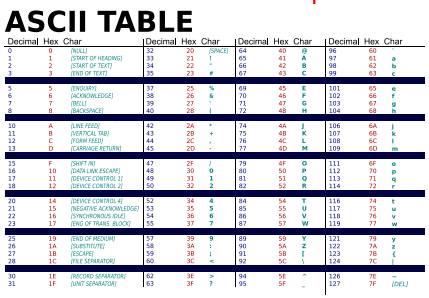
```
#include <stdio.h>

int main(void) {
  printf("size of char  = %d\n", sizeof(char));
  printf("size of short = %d\n", sizeof(short));
  printf("size of int   = %d\n", sizeof(int));
  printf("size of long  = %d\n", sizeof(long));
}
```

For printf() parameters, see for example C Reference Guide

```
size of char  = 1
size of short = 2
size of int   = 4
size of long  = 8
```

How is 'a' stored in memory?



Assignment

```
char c;
int i, j;

i = 10;
j = i;

c = i;

i = j + c;
```

What if i = 1024?

Compiler error or warning?

Compiler error or warning?

Value of i? Value of c?

Declarations and Assignments

```
#include <stdio.h>

int x;

int main()
{
    char y;

    x = 32;
    y = 'a';

    x = x + y;

    printf("x has now value %d and y has value %d, the code for character %c\n",
    return 0;
}
```

Declarations

Assignments

```
C18GLMM:mop ppedro$ ./x1
x = 293228598 y = 293228608
C18GLMM:mop ppedro$ ./x1
x = 251715638 y = 251715648
C18GLMM:mop ppedro$ ./x1
x = 59990070 y = 59990080
C18GLMM:mop ppedro$ ./x1
x = 157876278 y = 157876288
C18GLMM:mop ppedro$ ./x1
x = 157020214 y = 157020224
C18GLMM:mop ppedro$ ./x1
x = 129105974 y = 129105984
C18GLMM:mop ppedro$ ./x1
x = 405405750 y = 405405760
C18GLMM:mop ppedro$
```

What happens?

```
…
int x;
int y;
y = x + 10;
…
```

A declared variable which has not yet been assigned a value is called **uninitialized**

Type Conversion

```c
#include <stdio.h>

int x;

int main()
{
    char y;

    x = 32;
    y = 'a';

    x = x + y;

    printf("x has now value %d and y has value %d, the code for character %c\n", x, (int)y, y);
    return 0;
}
```

Implicit type conversion

Explicit type conversion

Type conversion is also called ***cast***, and you can say that you do ***casting***.

Arithmetic Operators

Basic assignment		a **=** b
Addition		a **+** b
Subtraction		a **-** b
Unary plus (integer promotion)		**+**a
Unary minus (additive inverse)		**-**a
Multiplication		a * b
Division		a **/** b
Modulo (integer remainder)		a **%** b
Increment	Prefix	**++**a
	Postfix	a**++**
Decrement	Prefix	**--**a
	Postfix	a**--**

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Arithmetic_operators

Comparison Operators

Equal to	a **==** b
Not equal to	a **!=** b
Greater than	a **>** b
Less than	a **<** b
Greater than or equal to	a **>=** b
Less than or equal to	a **<=** b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Comparison_operators.2Frelational_operators

Logical Operators

Logical negation (NOT)	**!**a
Logical AND	a **&&** b
Logical OR	a **||** b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Logical_operators

Compound Assignment Operators

Operator name	Syntax	Meaning
Addition assignment	a **+=** b	a **=** a **+** b
Subtraction assignment	a **-=** b	a **=** a **-** b
Multiplication assignment	a ***=** b	a **=** a ***** b
Division assignment	a **/=** b	a **=** a **/** b
Modulo assignment	a **%=** b	a **=** a **%** b
Bitwise AND assignment	a **&=** b	a **=** a **&** b
Bitwise OR assignment	a **|=** b	a **=** a **|** b
Bitwise XOR assignment	a **^=** b	a **=** a **^** b
Bitwise left shift assignment	a **<<=** b	a **=** a **<<** b
Bitwise right shift assignment	a **>>=** b	a **=** a **>>** b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Compound_assignment_operators

Bitwise Operators

Bitwise NOT	**~**a
Bitwise AND	a **&** b
Bitwise OR	a **|** b
Bitwise XOR	a **^** b
Bitwise left shift	a **<<** b
Bitwise right shift	a **>>** b

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Bitwise_operators

How we use Bitwise operations

- Bitwise OR is typically used to set certain bits.
- Bitwise AND is typically used to reset certain bits.
- Bitwise XOR is typically used to invert certain bits.
- Bitwise NOT used to invert all bits.

Quick Review:
- 0x (prefix for hexadecimal), 0b (prefix for binary)
- a << n (shift n bits of a to the left and fill with 0 bits coming from the right)
- a >> n (shift n bits of a to the right and fill with 0 bits coming from the left)

Expression evaluation

```
char c;
int i, j;

c = ’a’;
j = 1;
i = c & 0x01 + 512 * j++;
```

Conditionals: If-else statements

```
int x = -4;

if( x == 0 ){
    // ...
}

if( x ){
    // ...
}
else {
    // ...
}
```

Evaluates to false so if body does not execute

Evaluates to true so if body is executed (x != 0)

- **Zero** is considered **false**.
- Anything that is **not zero** is considered **true**.

Loops: while & for

```
int x = 5;

while( x!=0 )
    x--;
```

```
int x = 5;

while( x )
    x--;
```

```
int x;

for( x=5; x; )
    x--;
```

Three equivalent loops.

```
for(int i=0; i<5; i++ ) {
     if(…)
        continue; // jumps till next iteration
     if(…)
        break; // exists the loop.
     …
}
```

`break` and `continue`

Functions

```
#include <stdio.h>

int foo(int x, char y)
{
    int sum = 0;

    while(y > 0) {
        sum += x*y;
        y--;
    }

    return sum;
}
```

Arguments (or parameters)

Return value of return type

Arguments are ”pass-by value”

```
int var1;
char var2 = 7;
var1 = foo(5, var2);
```

var2 still has the value 7
after the function call

Function Prototype

```c
#include <stdio.h>

// function prototype
int foo(int x);

int main()
{
   printf("x is %d\n", foo(0));
   return 0;
}

int foo(int x)
{
   // function body
}
```

Where is the prototype for ***printf***?

Bitwise operations: Examples part 1

isSetLSB, isSetMSB, isSetN

```
int
isSetLSB( int x ) {
  if( x & 0x00000001 )
    return( 1 );
  else
    return( 0 );
}
```

```
int
isSetMSB( int x ) {
  return( x & 0x80000000 );
}
```

Is correct?

```
int
isSetMSB( int x ) {
  return( ( x & 0x80000000 ) == 0 ? 0 : 1 );
}
```

```
int
isSetN( int x, int n ) {
  int mask = 0x00000001;
  mask = mask << n;
  return( ( x & mask ) == 0 ? 0 : 1 );
}
```

```
NOTE:
0x – prefix for hexadecimal
0x0001 << 1 → 0x0002 (0000 0010)
0x0001 << 3 → 0x0008 (0000 1000)
0x0001 << 4 → 0x0010 (0001 0000)

0x0A52 = 0000 1010 0101 0010
0x0A52 << 2 → 0x2948 (0010 1001 0100 1000)
```

Bitwise operations: Examples part 2

countOnes, set mask bits (or reset mask bits)

```
int
countOnes( int x ) {
  int mask = 0x00000001;
  int count = 0;

  for( int i = 0; i < 32; i++ )
    if( x & (mask << i) ) count++;

  return count;
}
```

```
#define BIT0 0x00000001
#define BIT1 0x00000002
#define BIT2 0x00000004
#define BIT3 0x00000008
#define BIT4 0x00000010

…

int main( void ) {
  int mask = 0, value1, value2;
  …
  mask = BIT0 | BIT2 | BIT4;
  value1 = value1 | mask; //set
                                  //reset
}
```

Bitwise operations: Assignment!

Packing different values into a single variable:

- Pack and Unpack a date (DAY/MONTH/YEAR) into a word (integer) variable

Max day = 31 -> log2(31) = 4.9 -> 5 bits
Max month = 12 -> log2(12) = 3.6 -> 4 bits
Max year = 9999 -> log2(9999) = 13.3 -> 14 bits