


UNIVERSITA
DEGLI STUDI
FIRENZE

Software Engineering for Embedded Systems - A.A. 20/21


On structured development:
Structured Programming, Design, Analysis,
and back
old stuff from the 70s,
still crucial to plan and understand
what you are doing when you are coding with c ... and other languages



ENRICO VICARIO

Enrico Vicario
Dipartimento di Ingegneria dell'Informazione
Laboratorio Tecnologie del Software
Università di Firenze
enrico.vicario@unifi.it, stlab.dinfo.unifi.it/vicario


0



UNIVERSITA
DEGLI STUDI
FIRENZE


SW Engineering for
Embedded Systems 2020
Enrico Vicario

Part I: structured programming



1/44

1




UNIVERSITA
DEGLI STUDI
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario


the c language

- a c program specifies a computation through 3 constructs
 - variables
 - expressions
 - statements



2/44

2




UNIVERSITA
DIGITAL STUDY
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario

1/3 - Variables

- a Variable holds a Value of some Type
 - the value varies along the computation
 - the type remains unchanged
- a Type is a set of Values and a set of Operations
 - predefined basic types: int, float, char, ... with modifiers
 - pointers-to and arrays-of
 - user-defined types: struct (see later)
- a Variable has a Lifecycle
 - declaration
 - reference
 - (for user-defined types there will be also a Definition – see later)
- variable identity is resolved as the address

3




UNIVERSITA
DIGITAL STUDY
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario

Variables

- a Declaration introduces a Variable
 - a Type, a location in memory, a name
- Declaration applies not only to variables
 - semantic modifier
 - `int a;` // a is an int
 - `int* ptr;` // ptr is a pointer to an int
 - `int A[64];` // A is an array of 64 int
 - `int f(void);` // f is a function returning an int
 - the rule: from the name, first right and then left
- the point of Declaration identifies the context
 - *scope* of visibility of the name and *lifetime* of the declared entity

4



UNIVERSITA
DIGITAL STUDY
FIRENZE


SW Engineering for
Embedded Systems 2020
Enrico Vicario

Variables

- user defined types
 - are defined by aggregation of multiple declarations
 - with possible recursion through pointers


```
struct list {
    int value;
    struct list * next_ptr;
};
```
- a Reference identifies a (declared) Variable
 - by name: `int a;` ... a ...
 - by dereferencing a pointer (address): `int* ptr;` ... `*ptr...`
 - by arithmetic offset over an Array (address): `int A[64];` ... `A[13]...`
 - ... don't mess Variables and references to Variables

5



UNIVERSITA
 DEGLI STUDI
 FIRENZE

SW Engineering for
 Embedded Systems 2020
 Enrico Vicario


2/3 - Expressions

- an Expression
 - returns a value (in some Type) and produces side effects on Variables (semantics)
 - is specified through a repetitive combination of Variables and Constant Values through Operators (syntax)
- the two aspects of semantics are more or less intuitive

(e.g. x holds 2)	return value	side effects	remarks
x+3	5		
x<=3	1		no Boolean
x>=3	0		
x=3	3	x <- 3	= is an operator
x++ -	3	x <- 4	
++x	4	x <- 4	

- a good practice: don't mess side effects and returned values

6



UNIVERSITA
 DEGLI STUDI
 FIRENZE

SW Engineering for
 Embedded Systems 2020
 Enrico Vicario


Expressions - functions

- functions are a kind of Expression
 - in the semantics: they return a value and produce side effects
 - in the syntax: a combination of constants (the name) with values returned by Expressions (actual parameters) through an operator (.)
- in the syntax perspective, f(x,3) is somehow like x+3
 - but, returned value and side effects remain hidden
 - and, the operation semantics is user defined (which leads to function definition)

(e.g. x holds 2)	return value	side effects	remarks
x+3	5		
f(x,3)	hidden	hidden	

- (more on functions later)

7

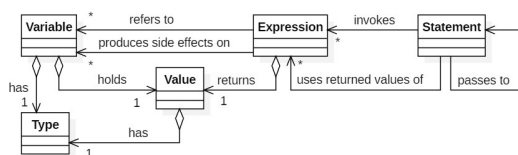


UNIVERSITA
 DEGLI STUDI
 FIRENZE

SW Engineering for
 Embedded Systems 2020
 Enrico Vicario

3/3 - Statements

- a Statement specifies
 - expressions to be evaluated
 - and the next statement to executed, possibly depending on values returned by expressions




- e.g.


```

int count; float A[64]; float sum;
for(count=0,sum=0;count<64;count++) sum+=A[count];
printf("«%f», sum);
      
```

8




UNIVERSITA
DIGITALI STUDI
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario

Statements

- Statements are
 - <Expr>; e.g. x=3;
 - Statement1 Statement2 e.g. x=3; y=x;
 - {Statement1 Statement2} e.g. {x=3; y=x;}
 - if(Expr)Statement
 - for(Expr1;Expr2;Expr3)Statement
 - while(Expr)Statement
 - do Statement while(Expr);
 - return Expr;
 - break
 - goto label
 - switch(Expr){case const: statement ...}
- Remark: statements and declarations have similar syntax




UNIVERSITA
DIGITALI STUDI
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario

9/44

9



UNIVERSITA
DIGITALI STUDI
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario

more on functions


- a function has a lifecycle
 - definition (being a kind of user defined operator)
 - declaration (to introduce its name in a context)
 - invocation (to let it be evaluated)
- a function *definition* specifies
 - returned type, name, formal parameter declarations list, body
 - the body may include among others:
 - local variable declarations, return statements, further function invocations

```
int add(int x, int y)
{
    int z;
    z=x+y;
    return z;}

```
- a function *declaration* (also knowns as prototype)
 - introduces the function name, returned type, and signature in a context (usually global), to make the function referrable

```
int add(int x, int y);

```




UNIVERSITA
DIGITALI STUDI
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario

10/80

10



UNIVERSITA
DIGITALI STUDI
FIRENZE


SW Engineering for
Embedded Systems 2020
Enrico Vicario

more on functions

- a function *invocation* is a kind of expression
 - made of the address where the function is stored followed by the list of actual parameters
 - the function address is usually specified by the declared name
 - actual parameters are expressions returning a value for each formal parameter in the definition

```
... add(x+3,y) ...

```
- on invocation and parameters binding
 - for each formal parameter, a variable is created (on the stack) and initialized with the value of the corresp. actual parameter (which is termed *binding by value*)
 - local variables declared in the body are allocated, and body statements are executed, until exhaustion of the code or reach of a return statement
- expression semantics
 - returned value: the value returned by the expression on return
 - side effects: those produced in the body execution



UNIVERSITA
DIGITALI STUDI
FIRENZE

SW Engineering for
Embedded Systems 2020
Enrico Vicario

11/44

11

UNIVERSITA' DIGITALI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

c language summary - 1/3

- A serialized story (a reasonable roadmap to learn the language)
 - Types, values, and constants
 - Variables
 - Expressions, side-effects and returned values
 - Pointers (a kind of variable)
 - Arrays (another kind of variable), static or dynamic allocation
 - Functions (a kind of expression), binding technique
 - Statements
 - structured types (a kind of user defined type)

12/44

12

UNIVERSITA' DIGITALI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

c language summary - 2/3

- The real story behind
 - Types and values
 - Elementary types,
 - user defined types (struct), type definition
 - Variables
 - Declaration and reference
 - Pointers, Arrays, static or dynamic allocation
 - Expressions
 - Operators, side-effects and returned values,
 - functions, binding technique, definition declaration and reference
 - Statements
 - Program structure
- A mechanism with 3 parties
 - Variables encode values (in types)
 - Expressions return a value and produce side effects on variables
 - Variables affect returned values and side-effects
 - Statements control the flow of execution of expressions
 - Expressions returned values affect statements

13/44

13

UNIVERSITA' DIGITALI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

c language summary - 3/3

- A mechanism with 3 parties
 - Variables encode values (in types)
 - Expressions return a value and produce side effects on variables
 - Variables affect returned values and side-effects
 - Statements control the flow of execution of expressions
 - Expressions returned values affect statements

```

classDiagram
    class Variable
    class Expression
    class Statement
    class Value
    class Type

    Variable "*" -- "*" Expression : refers to, produces side effects on
    Expression "*" -- "*" Statement : invokes, uses returned values of
    Statement "1" -- "*" Expression : passes to
    Variable "1" -- "*" Value : holds
    Value "1" -- "*" Type : has
    Expression "1" -- "*" Value : returns
  
```

14/44

14

UNIVERSITA' DIGITAL STUDY FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

Structured programming

- Is about the organization within the scope a function
- Basic principle: encode the state in the position of control
 - enables axiomatic reasoning (Floyd algorithm)
 - ... what you actually do while coding, ... and debugging
- Corollaries
 - no global variables
 - fulfill guard contracts in control statement
 - avoid goto, break, continue, return within a compound
 - separation of control and data variables
 - span of control
 - select proper cycling constructs (do, while, for)
 - according to whether the number of iterations is determined or not since the first iteration
 - ... and whether the guard is on entry or exit
 - no flag arguments passed-to/returned-by functions

15/44

15

UNIVERSITA' DIGITAL STUDY FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

Part II: structured design

16/44

16

UNIVERSITA' DIGITAL STUDY FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

A closer look at structuring constructs

- Functions as a means to give programs a structure
 - a kind of (dynamic) returning goto ...
 - ... with separated address spaces (variable lifetime / declaration scope)
 - 3 ways to support communication across separated address spaces
 - parameters binding technique
 - by value (actual parameters are expressions), using pointer values to produce side-effects
 - by reference (actual parameters are references)(not in c)
 - returned value (function calls are a kind of expression)
 - global variables):
- Structured user defined types (struct)
 - aggregation of cohesive variables enforced by the language
 - reduces complexity of interfaces
 - is a concept orthogonal to functions
 - (but, the balance between function and data is not symmetric, in c)

17/44

17

Structured design

- Is about the organization of functions (interprocedural)
 - Allocation of responsibilities to modules
 - Hierarchical caller/callee relation among modules
- abstracts from the procedure
 - ... to emphasize the hierarchical relations among functions
- function interfaces are a secondary concept
 - more generally, data are decorations on functions
- relies on a modeling artifact: the *structure chart*
- according to a methodology: based on *coupling* and *cohesion* metrics

18

The modeling artifact: Structure Chart - 1/4

- The structure chart is a tree (a graph) made of modules and couples
- Modules are the functions

19/44

19

The modeling artifact: Structure Chart - 2/4

- Couples are the parameters bound at function call
 - Data vs control couples
 - Direction: input or output (i.e. side-effected)
 - Idiom: pass the address by value to produce a side-effect
 - Shared variables
 - a global variable, a database or file record, the network

20/44

20

The modeling artifact: Structure Chart - 3/4

- Procedural annotations
 - repetition (while, do, for)
 - guard (if, switch)
 - One shot call (a marginal idiom based on the static keyword)

- I/O
- ... and much more
 - Macro, asynch, coroutine ...
- From the '70, not explicitly intended for the c language

21/44

21

A metrics of evil: coupling - 1/3

- Coupling Is about how much you must know about module X in order to safely modify module Y
- Is a metrics of dis-quality
 - hurdles maintenance and separated evolution of modules
- pushes towards integrating modules
 - Topological partitionining, aka refactoring

22

A metrics of evil: coupling - 2/3

- Coupling derives from various factors
- Function calls
 - Coupling grows with the complexity of the interface
 - Control couples are worse than data couples, especially in the direction from callee to caller
- Common areas
 - Also a file, a db record, an interacting application
- Pathological references
 - Not explicitly supported in c, but easily (and often) produced using pointers
- Coupling decreases with the binding time
 - coupling at compile time is worse than at link-time,
 - and at link-time is worse than at run-time

23

UNIVERSITA' DIGLI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

A metrics of evil: coupling - 3/3

- Coupling is transitive
 - Though it gets discounted
 - Interfaces may serve to decouple

24/44

24

UNIVERSITA' DIGLI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

A metrics of good: cohesion - 1/3

- Cohesion is about how much two responsibilities in the same module are cohesive to each other
 - Including the hierarchy under the module
- Is a metrics of quality
 - Cohesive responsibilities will evolve together, and will be modified by people with homogeneous domain
 - Avoid partial maintenance
 - Reduce the number of reasons for modifying a module
 - Promote reuse

25/44

25

UNIVERSITA' DIGLI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

A metrics of good: cohesion - 2/3

- frequently (ab)used (pathological) criteria of cohesion
- Coincidental: 0
 - E.g. 2 lines of code that frequently appear close to each other
 - Typical when retro-fitting a flat design into a hierarchy
- Logical: 1
 - Similar operations
 - E.g. input of network parameters and data to be processed
- Temporal: 3
 - Operations executed at close time instants
 - E.g. initialization of a video card or a network card
 - N.B.: the distance in temporal projection of a procedure does not necessarily follow from the distance in code structure
- ... (continues)

26/44

26

UNIVERSITA' DIGITALI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

- **Procedural: 5**
 - Steps bound together by the structure of procedure
 - E.g. 2 statements under the same guard
 - Typically occurring when retrofitting a flow-chart design into a hierarchy

27/54

27

UNIVERSITA' DIGITALI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

A metrics of good: cohesion - 3/3

- virtuous criteria of cohesion (at least for structured design)
- **Communicational: 7**
 - Two modules operating on the same data elements
- **Sequential: 9**
 - Two modules in a pipe: the former produces data for the latter
- **Functional: 10**
 - Two functions that are both essential for another function
 - Good example: the scooter keys and the helmet
 - Bad example: the egg and the pan

<TBD: trovare un esempio meno metaforico!>

28/44

28

UNIVERSITA' DIGITALI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

Example: selling flight tickets

- **Functional requirements**
 - Receives date, destination, customer name, price rates, flights, and available places
 - Books and reserves a place, and prints a ticket
 - While so doing, applies a pricing policy based on the customer history.

29/44

29

UNIVERSITA' DEGLI STUDI FIRENZE SW Engineering for Embedded Systems 2020 Enrico Vicario

Example: selling flight tickets

- A possible design of the structure
 - check cohesion of functions allocated to modules in the same sub-tree
 - ... and coupling among different modules

```

graph TD
    reserve_flight[reserve_flight()] --> getFlightWithPrice[getFlightWithPrice()]
    reserve_flight --> selectFlight[selectFlight()]
    reserve_flight --> reserve[reserve()]
    getFlightWithPrice --> getFlights[getFlights()]
    getFlightWithPrice --> getFlightWithPlaces[getFlightWithPlaces()]
    getFlights --> getDestAndDate[getDestAndDate()]
    selectFlight --> getBasePrice[getBasePrice()]
    reserve --> commitReservation[commitReservation()]
    reserve --> emitTicket[emitTicket()]
    reserve --> updateCustomerFidelity[updateCustomerFidelity()]
    getBasePrice --> getBasePrice2[getBasePrice()]
    getBasePrice --> getCustomerFidelity[getCustomerFidelity()]
    getBasePrice2 --> getBasePrice3[getBasePrice()]
    getBasePrice2 --> getBasePrice4[getBasePrice()]
  
```

- ... cohesion and coupling are about evaluating whether this can fit
 - but, how was all this sorted out?

30/44

30
