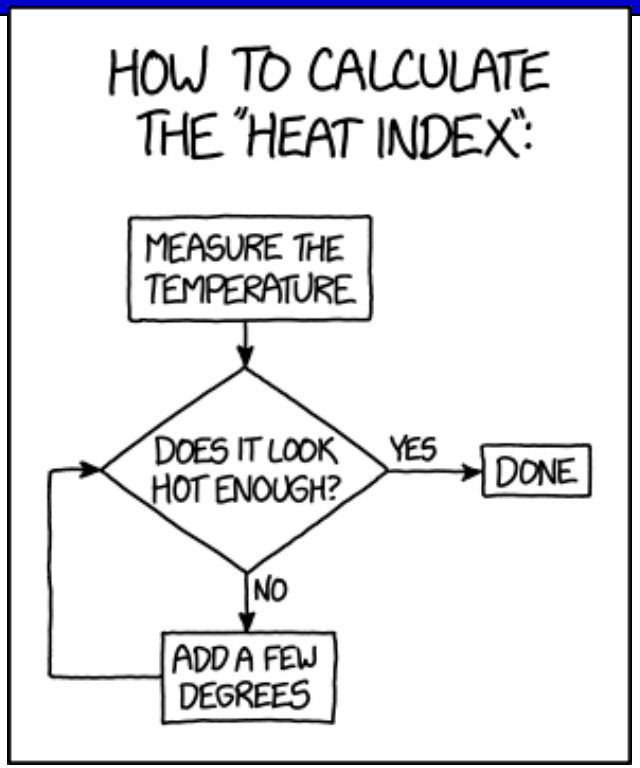
C Programming I: Lecture II

Andreas Heinz
FYD400 - HT 2019



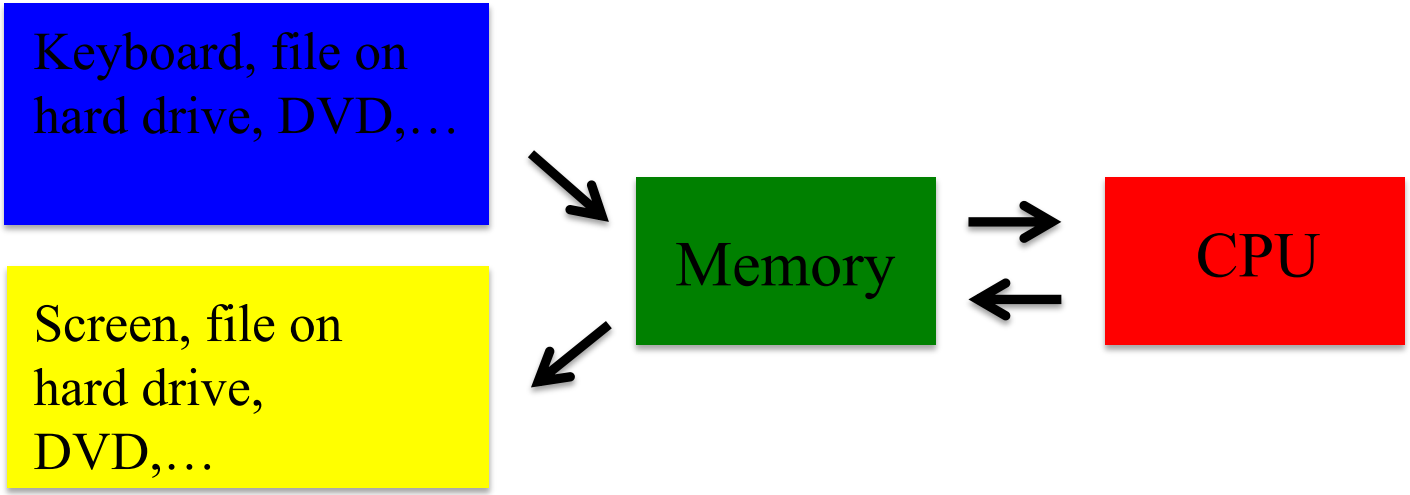
xkcd.com

"I probably remember 20% of the stuff
I learned in school and forgot the other 90%."

RECAP

Computers

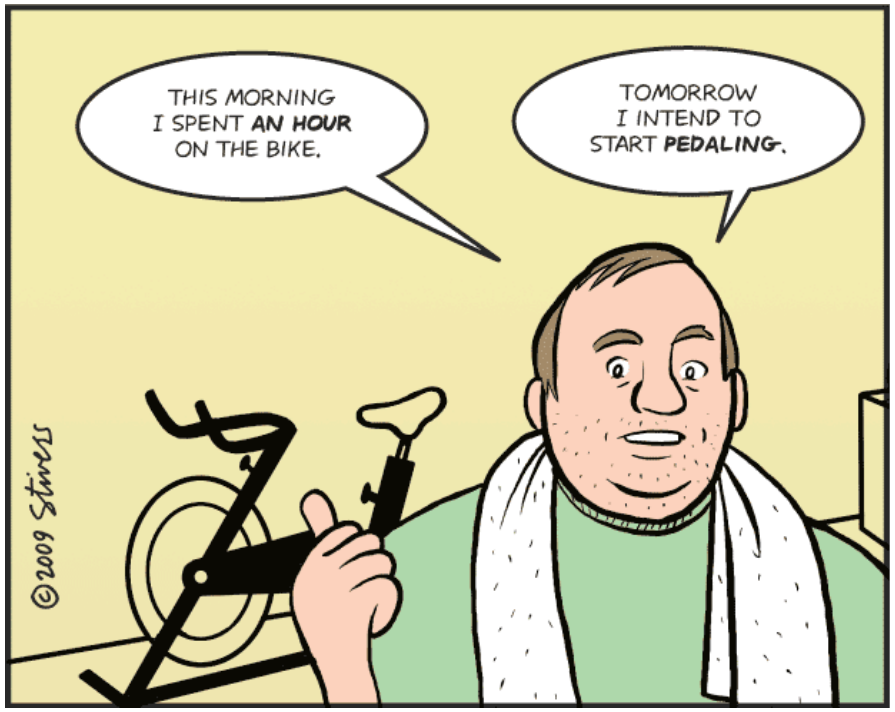
- "Definition" of a computer:
  - A device which can manipulate data according to the instructions of a program.
  - Early programming: mechanical, then wires, relays and transistors
  - Now: integrated circuits
  - von Neumann: program and data are both stored in the memory



Warm-up

Take a piece of paper and write down the C code for the "Hello World!" program we discussed in the last lecture.

This is not an exam and no one but you will see what you wrote.



C: "Hello World!"

Create a file, e.g. code.c in an editor – this is your source code

```
/* printing program */
#include <stdio.h>

int main (void) // sometimes just int main ()
{
        printf ("Hello\n World!\n");      /* that's correct */
        return 0;
}
```

To standard output:

```
Hello
World!
```

Note: C is case sensitive!

How to Create a Program?

```
#include <stdio.h>
int main(void)
{
        printf("Hello World!");
        return 0;
}
```

IDE

Source code: file.c

Compiler

```
00010011100010101000110000011110 0
01110010101000111100101010010010 1
```

Object file: file.obj

Library

Linker

```
00010011100010101000110000011110 0
01110010101000111100101010010010 1
10000110001111111110011111101011 0
01001100011110101010100111101010 11
```

Program (executable file): file.exe

Lecture 2 Contents

- **Vocabulary**
- **Algorithms**
- **Types**
- **Compilation**

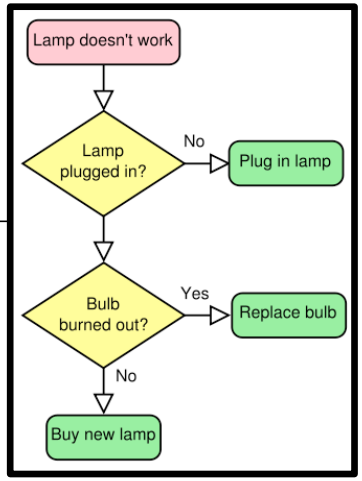
Vocabulary

- **Source code, (preprocessor), compiler, linker, executable (code.exe [only in windows])**
- **Include file (.h), standard library, preprocessor directive**
- **main function, there can be only one, start of the execution**
- **Function: return type, parameter, header, body**
- **Statement: compound statement/block, forgetting ";" – the most common mistake**
- **"text string" (string literal), escape sequences e.g. "\n"**
- **Return value (max. 1 per function), reserved words/keywords (table in VtC)**
- **Comments /* … */ versus //**

Back to Programming in General

What is an algorithm?

Programming: Choice of an Algorithm

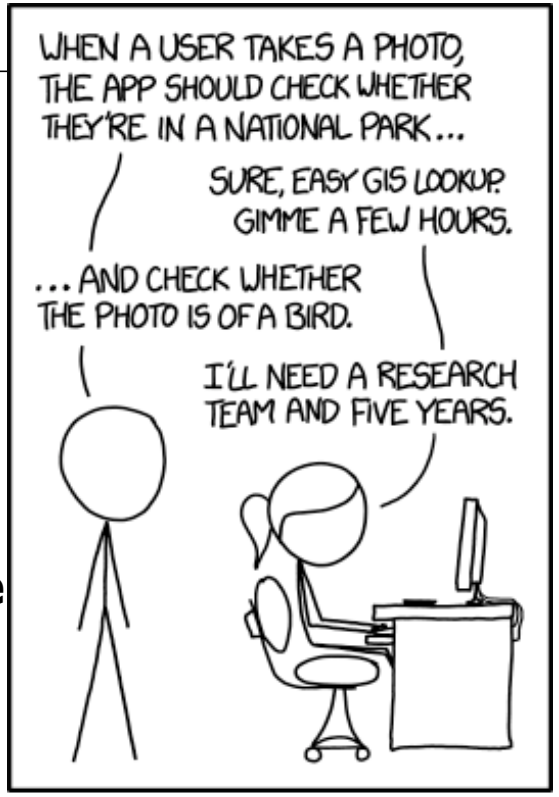


- **Goal**:
  - solution of the problem
  - independent of programming language
  - logic
- **Tools** (similar in most programming languages but often with different names):
  - Assignment/initialize
  - Call
  - IF…THEN…ELSE
  - Increment
  - Module
  - Loop
  - OPEN/CLOSE file
  - READ
  - WRITE

There are several ways to solve a programming task! Even small changes in the specification may require a new algorithm!

Programming: Choice of an Algorithm

- **Goal**:
  - **solution of the problem**
  - **independent of programming language**
  - **logic**
- **Tools (similar in most programming languages but often with different names):**
  - **Assignment/initialize**
  - **Call**
  - **IF…THEN…ELSE**
  - **Increment**
  - **Module**
  - **Loop**
  - **OPEN/CLOSE file**
  - **READ**
  - **WRITE**



xkcd.com

There are several ways to solve a programming task! Even small changes in the specification may require a new algorithm!

Stepwise refinement

Does the program solve the task ? ->
Refine

- **break down into smaller parts - in steps**
- **How many resources (memory, time) does each step take?**
- **Flexibility?**
- **Secure?**
- **Clarity (comments!)?**
- **Do I need another algorithm?**
- **"Occam's razor" – restrict yourself to the necessary**

Example: Opening a door

- **Grip door handle**
  - **Lift arm**
  - **Put hand on door handle**
  - **Bend fingers around it**
  - **Press fingers against handle**
- **Push door handle down**
- **Push door open**

The Most Important Task of a Program

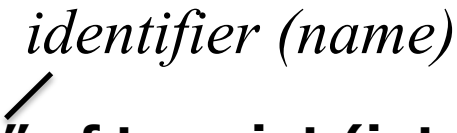
Manipulate Data!

⇒ need to store data temporarily

⇒ variables

C: Data Types

- C is a statically typed language – apples in the apples box…
- A variable has a type and a "name" coupled together to form an "object" (= memory space)



- *int a;* // declaration of variable "a" of type int (integer)
- *int a=5;* // declaration and assignment – gives the variable a value, initialization
- Size (in memory) of the object depends on the type
- Some basic types in C // there are more (lectures 3 and 4)
  - Integers
  - Decimals / floating point numbers
  - Characters
  - *void* (generic memory address)

C: Integer Types

bit = 0,1
1 byte = 8 bit: 0 - 255

- **integers: short int, int, long int, long long int; unsigned or signed (default)**
- **8 (char), 16, 32, 64 bits // actual limits in limits.h - problem with portability (see C99 in chapter 12)**
- **use the right size, don't waste memory, know your operating system**
- **decimal, octal, hexadecimal number systems**

Examples: 13, 017, 0xf

the first digit is a "zero"

C: Floating Point Numbers

- Examples: 114.15, 0.11415E3, 22323.3f
- sign, fraction and exponent / / float.h, see limits.h
- rounding -> finite precision!
- float (32 bit), double (64 bit), long double (128 bit)
- float: precision about six decimal places, $\approx 1.2 \times 10^{-38}$ – $\approx 3.4 \times 10^{38}$
- double: $\approx 2.2 \times 10^{-308}$ – $\approx 1.8 \times 10^{308}$
- floating-point types are "higher" types than integers (VtC, chapter 3.6)

C: Type Conversion (casting)

- **automatic (implicit) or "manual" (explicit) "casting"**
- **automatic:**

Note that the type of b remains unchanged in subsequent statements!

```
int a;
double b=3.7;
a = b; // assignment: b is casted to type int ,
       // a becomes 3
float d, c = 5.3;
d = c * a; // operand with the highest type
           //  determines type of the right-
           // hand side; d becomes 15.9
```

- **explicit: use typecast - (type) expression // use carefully!**

Example:

```
int e, g;
e = (int) b; // e is of type int and has value 3
long double f = 4.18478E14;
g = (int) f; // (int)f is not defined!
```

See chapter 3.6!

Example

```
/* Calculate the worth of your weight in platinum */
#include <stdio.h>
int main(void) {
   float weight;   /* user weight              */
   float value;    /* platinum equivalent     */

   printf("Are you worth your weight in platinum?\n");
   printf("Let's check it out.\n");
   printf("Please enter your weight in kg: ");

   /* get input from the user */
   scanf("%f", &weight);
   /* assume platinum price is 8000 SEK per ounce; 32.1507466 converts kg to ounces (troy)  */
   value = 8000.0f * weight * 32.1507466f;                    /* is this correct?                    */
   printf("Your weight in platinum is worth $%.2f.\n", value); /* %.2f is a conversion specifier */
   printf("You are easily worth that! If platinum prices drop,\n");
   printf("eat more to maintain your value.\n");

   return 0;   }
```

C: Characters and Identifiers

In C: "One byte is the number of bits needed to store a char."

- **Characters**
  - **ASCII, 128 characters (7 bit)**
  - **A-Z, a-z, control sequences, and symbols fill the 7 bit**
  - **åäö may be found in LATIN-1 (256 characters based on ASCII).**
  - **Treated by C as small integers**
- **Identifiers**
  - **"name" of functions, variables, MACROs**
  - **"A-Z", "a-z", "0-9" (not by themselves!), "_", no keywords, (no åäö)**
  - **Good examples: "a", "read_text", "money_temp", "TEST"**
  - **Bad examples: "gkhdfkjkt", "_hel", "hT" and "Ht", "printf"**
  - **Case sensitive**
- **More on that in lecture 3 (storage classes)**

ASCII Table

Dec	Hx	Char	Dec	Hx	HTML	Char	Dec	Hx	HTML	Char	Dec	Hx	HTML	Char
0	0	NUL (null)	32	20	&#32;	Space	64	40	&#64;	@	96	60	&#96;	`
1	1	SOH (Start of heading)	33	21	&#33;	!	65	41	&#65;	A	97	61	&#97;	a
2	2	STX (Start of text)	34	22	&#34;	"	66	42	&#66;	B	98	62	&#98;	b
3	3	ETX (End of text)	35	23	&#35;	#	67	43	&#67;	C	99	63	&#99;	c
4	4	EOT (End of transmission)	36	24	&#36;	$	68	44	&#68;	D	100	64	&#100;	d
5	5	ENQ (Enquiry)	37	25	&#37;	%	69	45	&#69;	E	101	65	&#101;	e
6	6	ACK (Acknowledge)	38	26	&#38;	&	70	46	&#70;	F	102	66	&#102;	f
7	7	BEL (Bell)	39	27	&#39;	'	71	47	&#71;	G	103	67	&#103;	g
8	8	BS (Backspace)	40	28	&#40;	(	72	48	&#72;	H	104	68	&#104;	h
9	9	TAB (Horizontal tab)	41	29	&#41;	)	73	49	&#73;	I	105	69	&#105;	i
10	A	LF (NL line fd, new line)	42	2A	&#42;	*	74	4A	&#74;	J	106	6A	&#106;	j
11	B	VT (Vertical tab)	43	2B	&#43;	+	75	4B	&#75;	K	107	6B	&#107;	k
12	C	FF (NP form fd, new page)	44	2C	&#44;	,	76	4C	&#76;	L	108	6C	&#108;	l
13	D	CR (Carriage return)	45	2D	&#45;	-	77	4D	&#77;	M	109	6D	&#109;	m
14	E	SO (Shift out)	46	2E	&#46;	.	78	4E	&#78;	N	110	6E	&#110;	n
15	F	SI (Shift in)	47	2F	&#47;	/	79	4F	&#79;	O	111	6F	&#111;	o
16	10	DLE (Data link escape)	48	30	&#48;	0	80	50	&#80;	P	112	70	&#112;	p
17	11	DC1 (Device control 1)	49	31	&#49;	1	81	51	&#81;	Q	113	71	&#113;	q
18	12	DC2 (Device control 2)	50	32	&#50;	2	82	52	&#82;	R	114	72	&#114;	r
19	13	DC3 (Device control 3)	51	33	&#51;	3	83	53	&#83;	S	115	73	&#115;	s
20	14	DC4 (Device control 4)	52	34	&#52;	4	84	54	&#84;	T	116	74	&#116;	t
21	15	NAK (Negative acknowledge)	53	35	&#53;	5	85	55	&#85;	U	117	75	&#117;	u
22	16	SYN (Synchronous idle)	54	36	&#54;	6	86	56	&#86;	V	118	76	&#118;	v
23	17	ETB (End of trans. block)	55	37	&#55;	7	87	57	&#87;	W	119	77	&#119;	w
24	18	CAN (Cancel)	56	38	&#56;	8	88	58	&#88;	X	120	78	&#120;	x
25	19	EM (End of medium)	57	39	&#57;	9	89	59	&#89;	Y	121	79	&#121;	y
26	1A	SUB (Substitute)	58	3A	&#58;	:	90	5A	&#90;	Z	122	7A	&#122;	z
27	1B	ESC (Escape)	59	3B	&#59;	;	91	5B	&#91;	[	123	7B	&#123;	{
28	1C	FS (File separator)	60	3C	&#60;	<	92	5C	&#92;	\	124	7C	&#124;	|
29	1D	GS (Group separator)	61	3D	&#61;	=	93	5D	&#93;	]	125	7D	&#125;	}
30	1E	RS (Record separator)	62	3E	&#62;	>	94	5E	&#94;	^	126	7E	&#126;	~
31	1F	US (Unit separator)	63	3F	&#63;	?	95	5F	&#95;	_	127	7F	&#127;	DEL

Char and int – an example

```
#include <stdio.h>

int main(void)
{
        int i = 17;
        char c = 'c';                   // ascii value is 99
        int sum;

        sum = i + c;                    // not possible in most languages
        printf("Value of sum : %d\n", sum );
        return 0;
}
```

C: Enumerated Type (enum)

- **create a "casual list"; limiting the number of choices, if variable can have only a small set of meaningful values**
- **in ANSI C**
- ***enum identifer { };***

Example:

```
enum subject {physics, chemistry, mathematics}; // repeated declaration
enum subject lecture, lab1, lab2;  // 3 variables of type "enum subject"

lab1 = chemistry;
lab1 = 1; //  same as above – compiler assigns number to the elements
lab 1 = 3; // Error! – only 0, 1, 2 work
lab 1 = exam; // Error!
```

C: Void

- **void = "empty"**
- **don't declare variables with *void* or give values of a specific type to an identifier of type *void***
- **functions: parameter or return values are missing**

  Example: *void fargfix();* // return value is void, i.e. empty
  int help (void); // no parameter in function call

- **more on type void in connection with pointers and functions in lectures 3 -4.**

C: Preprocessor

- a dumb text editor or a very efficient tool
- includes files (.h)
  - provides access to functions from standard libraries
  - write own .h files (see chapter 9.3), <name.h> vs. “name.h”
  - allows more structured programming (lecture 3)
- conditional compiling
  - platform-dependent programs
  - many versions of the same program

  Example: see chapter 9.4
- global replacement of text => MACROs…

C: Preprocessor: Macro

```
#define BASE 50 // something that rarely changes, but has no type!
#include <stdio.h>

int main (void)
{
        float number, price;
        printf ("Number of kWh?\n"); // can change at any time
        scanf ("%f", &number);
        printf ("Price of kWh?\n");      // also this can change
        scanf("%f", &price);
        printf("Bill: %.2f", (number * price) + BASE);
        return 0;
}
```

MACRO BASE initialized with value 50

C: const. vs. macro

```
/*# define BASE 50 */ // changes rarely; easy to edit
#include <stdio.h>

int main (void)
{
        const int base = 50; // constant of type int instead of macro
        float number, price;
        printf ("Number of kWh?\n"); // can change at any time
        scanf ("%f", &number);
        printf ("Price of kWh?\n");     // also this can change
        scanf("%f", &price);
        printf("Bill: %.2f", (number * price) + base);
        return 0;
}
```

C: Preprocessor


```
/* Conversion Fahrenheit -> Celsius */
#include <stdio.h>
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f /9.0f)

int main (void)
{
float fahrenheit, celsius
printf (“Enter Fahrenheit: “);
scanf (“%f”, &fahrenheit);
celsius = (fahrenheit – FREEZING_PT) *
SCALE_FACTOR;
printf(“Celsius: %.1f\n”, celsius);
return 0;
}
```

```
blank line
lines brought in from stdio.h
blank line
blank line
blank line
int main (void)
{
float fahrenheit, celsius
printf (“Enter Fahrenheit: “);
scanf (“%f”, &fahrenheit);
celsius = (fahrenheit – 32.0f) * (5.0f
/9.0f);
printf(“Celsius: %.1f\n”, celsius);
return 0;
}
```

Preprocessor: Example

Conditional compiling:

in e.g. operational.h // header file (.h):

```
/* operational.h */

# define windows7
```

in program.c // your source code:

```
# include “operational.h”
# ifdef windows7
        #include “win.h” // note: not <name.h> but “name.h” because it is
                         // located in your working directory
#endif
                         // provides access to win.h if the OS is windows 7
                         // “downward compatibility” – important element to
                         // consider in programming
int main (void) { …
```

Compiling

source code ("program text") in C	**filename.c**	**// plain text**
C complier		**// LW CVI**
machine language ("object file")	**filename.obj**	**// system specific**
+ (standard) functions (from library)	**filename.h**	**// built-in or made**
		// by you
Linker		**// static or dynamic**
executable program	**file.exe**	**// final "product"**
***OS* (= Operating System)**		**// e.g. Windows 10**

program gets loaded into memory and executed

Note: object file extension can be different for different compilers/OS; e.g. ".o" for gcc.

How to Create a Program?

```
#include <stdio.h>
int main(void)
{
        printf("Hello World!");
        return 0;
}
```

IDE

Source code: file.c

Compiler

```
00010011100010101000110000011110 0
01110010101000111100101010010010 1
```

Object file: file.obj

Library

Linker

```
00010011100010101000110000011110 0
01110010101000111100101010010010 1
10000110001111111110011111101011 0
01001100011110101010100111101010 11
```

Program (executable file): file.exe

Summary of Lecture 2

- **Recap – repetition is useful**
- **Vocabulary – you need to speak the speak**
- **Algorithms – this is of the hardest part of programming**
- **Data types – integers, floats, chars – the way we store data**
- **Preprocessor directives – widely used in C**