Programming Project (2)

Yesterday, we looked at:

- Programming Process
- Basic structure of a C program
- Data Types in C
- Escape sequences
- Declaration & Initialization
- C operators
- Type Converting / Type Casting
- `printf()`: formatted output to a device
- Control Flow
- Exercise 1: HW, Compiler, Debug, Exe, ANSI escape codes
- Exercise 2: Structural C

Today, we will look at:

- Control Flow (cont.): if, switch, while, do, for, break & continue, goto
- Range & precedence of operators
- Program structure in C
- Software Architecture & Documentation
- Project Schedule & Block Diagram
- Tables & Arrays. Arrays & Pointers
- Pointers and Function Arguments
- Structures
- Typical errors in C
- Exercise 3: Fixed Point Arithmetic
- Exercise 4:
- Bit Manipulation Exercise

Control Flow (repetition)

Statements

```
i=a+b/j;
```

Block Statements

```
{
 i=0;
 j=5*i+2;
}
```

If-else Statements

```
if (udtryk) statement;

if (udtryk) statement1;
else statement2;

if (udtryk1)
     if (udtryk2) statement1;
     else statement2;

if (udtryk1) statement1;
else if  (udtryk2) statement2;
else if  (udtryk3) statement3;
else statement4;
```

Switch

```
switch (udtryk)
    {
     case konstantudtryk1: statement1; break;
     case konstantudtryk2: statement2; break;
     default: statement3; break;
    }
```

While

```
while (udtryk) statement;
```

Do while loop

```
do statement;
while (udtryk);
```

For loop

```
for (udtryk1a, udtryk1b; udtryk1b; udtryk3) statement;

udtryk1;
while (udtryk2)
    {
     statement;
     udtryk3;
     }
```

Comma operator (in for): `udtryk1 : j=0, i=5`

If-else Example

```
#include "30010_io.h"
#include <sio.h>

void main()
   {
    char t1='8',t2=-'B';
    int cif,tal1,tal2,max,flag=-1;
    ... ... ...
    if (t1>='0' && t2<='9') cif=t1-'0';

    if (t2>='0' && t2<='9') cif=t2-'0';
    else if (t2>='A' && t2<='F') cif=t2-'A'+10;
    else cif=-1;

    printf ("\n Indtast to heltal: ");
    scanf ("%d %d", &tal1, &tal2); // read from In
    if (tal1==tal2) printf ("\nEns tal");
    else
       {
        max=(tal1>tal2) ? tal1 : tal2;
        printf ("\nForskellige tal, max %d",max);
        if (max>=10)
           if (max==0) flag=0;
           else flag=1;
        printf ("\n Flag= %d",flag);
       }
 ... ... ...
   }
```

Switch Example

```
#include "30010_io.h"
#include <sio.h>

void main()
   {
    char dag;

    dag = getch(); // read from in

    switch (dag)
    {
     case '1': printf ("Mandag\n");
               break;
     case ‘2': printf (“Tirsdag\n");
               break;
     case ‘3': printf (“Onsdag\n");
               break;
     case ‘4': printf (“Torsdag\n");
               break;
     case ‘5': printf (“Fredag\n");
               break;
     case ‘6': printf (“Lørdag -");
               break;
     case ‘7': printf (“Weekend\n");
               break;
     default : printf (“Fejl\n");
               break;
    }
   }
```

While Example

```
#define BLANK 0x20

void main()
   {
    char ch,s[]=‶   TESKST″;
    int i=0, blank=0, tegn=0;

    while (s[i]==BLANK) i++;

    i=0;
    while (s[i++]==BLANK);

    i=0;
    while ((ch=s[i0++] !=´\0´)
       {
        if (ch==BLANK) blank++;
        else tegn++;
       }
    }
```

Do Example

```
void main()
   {
    int tal=12345,cif=0;

    if (tal<0) tal=-tal;
    do
       {
        cif++;
        tal=tal/10;
       }
    while (tal!=0);
     }
```

For Example

```
#define FALSE 0
#define TRUE  1
#define MAX   10

#include <eZ8.h>
#include <sio.h>

void main()
   {
    char slut=FALSE;
    int i,j;
    float tal[MAX],sum;

    for (i=0; !slut && i<MAX; i++)
    {
     printf ("\n Indtast kommatal: ");
     scanf (‶%f‶,&tal[i]);
     if (tal[i]==0.0 slut=TRUE);
    }

    sum=0.0;
    for (j=0; j<1; j++) sum+=tal[j];
    for (j=0, sum=0.0; j<1; sum+=tal[j++]);
   }
```

Goto Example

```
... ... ... ... ... ...
if (fejlsituation) goto fejl;
... ... ... ... ... ...
... ... ... ... ... ...
fejl:
//Fejlbehandling
```

Range & precedence

Operator	Associativity
(*expr*) [*index*] -> .	Left ==> Right
! ~ ++ -- (*type*) sizeof “Unary operator” + - * &	Right ==> Left
* / %	Left ==> Right
+ -	Left ==> Right
<< >>	Left ==> right
< <= > >=	Left ==> Right
== !=	Left ==> Right
“Binary operator” &	Left ==> Right
“Binary operator” ^	Left ==> Right
“Binary operator” |	Left ==> Right
&&	Left ==> Right
||	Left ==> Right
expr ? true_expr : false_expr	Right ==> Left
+= -= *= /= <<= &= ^= |= %= >>= =	Right ==> Left
,	Left ==> Right

Associativity gives the direction of evaluation for operators with the same priority. Fx:

a/b*c = (a/b)*c

Operators in the 1st line have the highest priority
Operators in the 2nd line have the next highest priority
And so on …

Fx: ++a->b = ++(a->b)

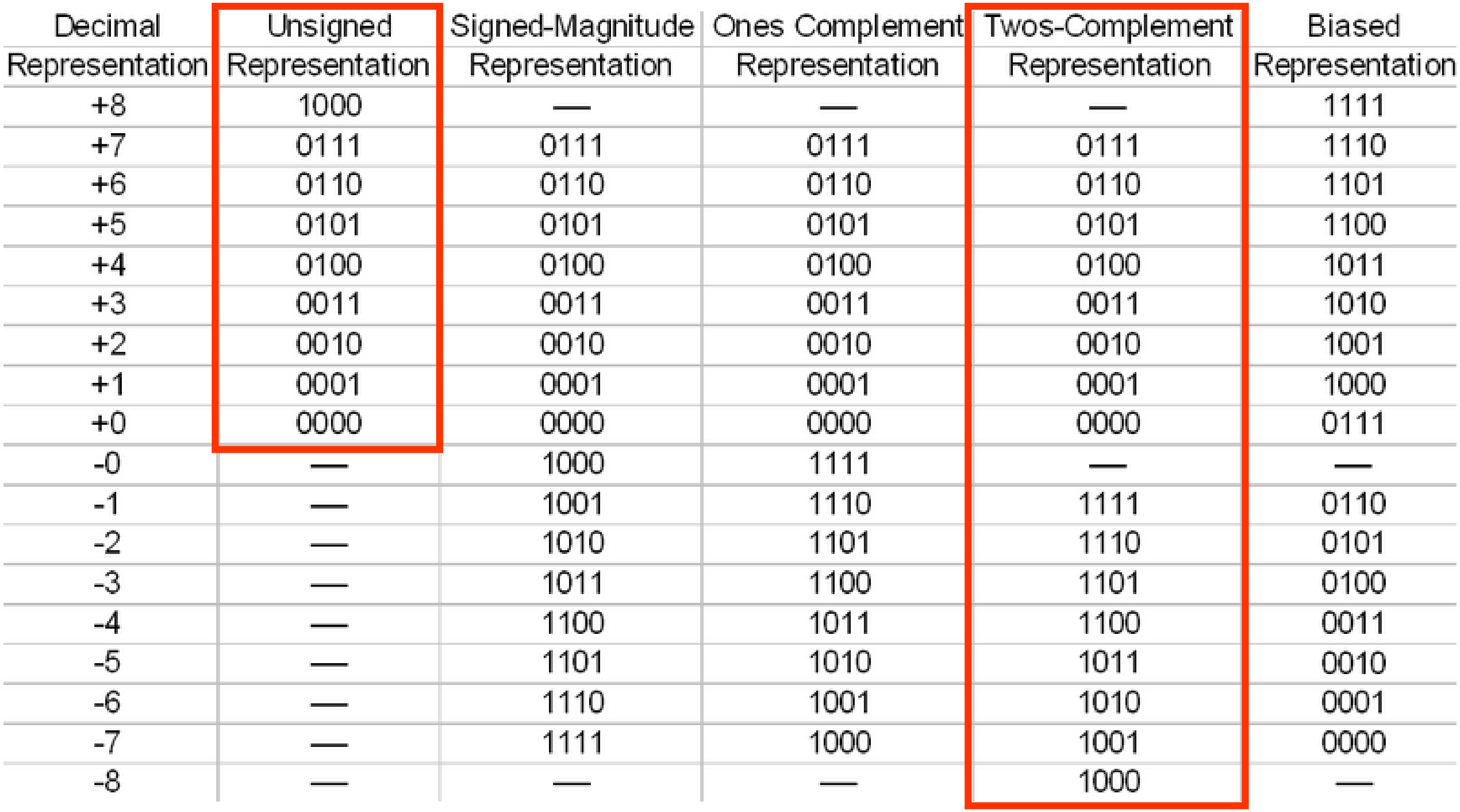
Unary operator	Example
+	+23209
-	-value
*	*pointer
&	&variable
Binary operator	**Example**

&	terrance = 0xCC; phillip = 0xAA;
	(terrance & phillip) == 0x88;

^	right = 0xF0; wrong = 0xCC;
	(right ^ wrong) == 0x3C;

|	curds = 0x99; whey = 0x96;
	(curds | whey) == 0x9F;

Integer Number Representation



Decimal Representation	Unsigned Representation	Signed-Magnitude Representation	Ones Complement Representation	Twos-Complement Representation	Biased Representation
+8	1000	—	—	—	1111
+7	0111	0111	0111	0111	1110
+6	0110	0110	0110	0110	1101
+5	0101	0101	0101	0101	1100
+4	0100	0100	0100	0100	1011
+3	0011	0011	0011	0011	1010
+2	0010	0010	0010	0010	1001
+1	0001	0001	0001	0001	1000
+0	0000	0000	0000	0000	0111
-0	—	1000	1111	—	—
-1	—	1001	1110	1111	0110
-2	—	1010	1101	1110	0101
-3	—	1011	1100	1101	0100
-4	—	1100	1011	1100	0011
-5	—	1101	1010	1011	0010
-6	—	1110	1001	1010	0001
-7	—	1111	1000	1001	0000
-8	—	—	—	1000	—

Source: http://en.wikipedia.org/wiki/Signed_number_representations

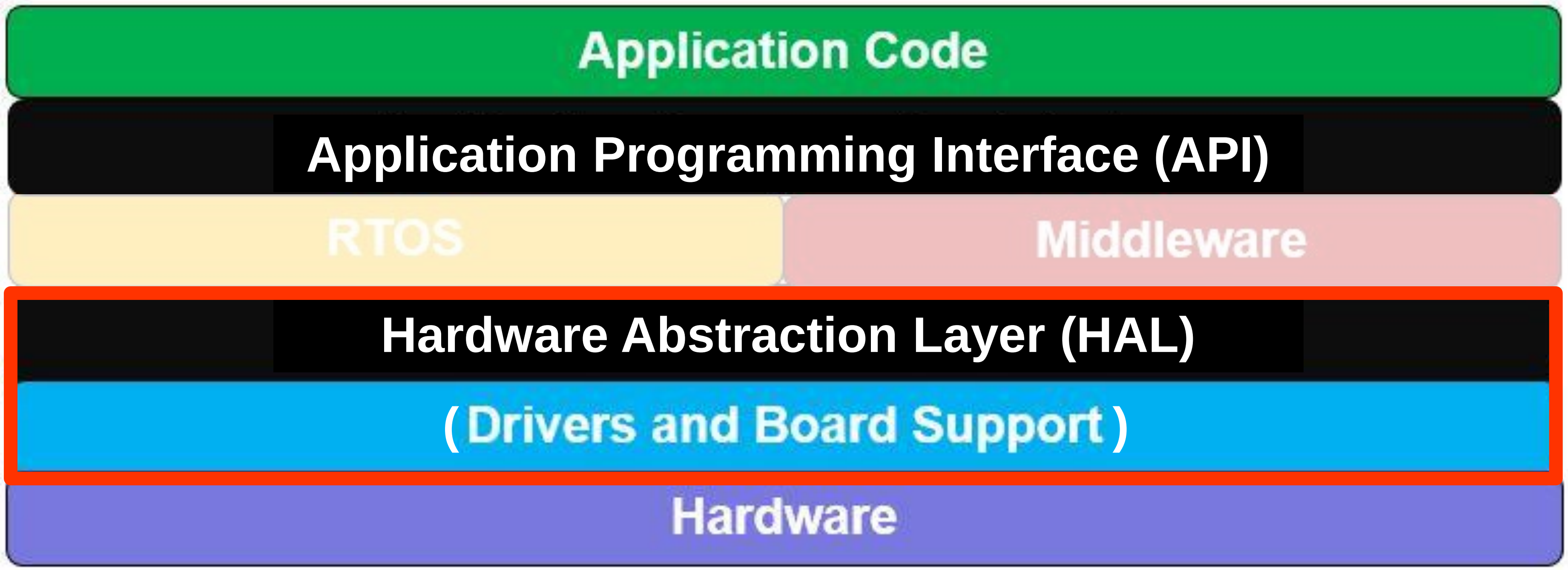
Negative numbers in 2's complement are formed as the inverted of the magnitude in positive +1 in binary

Bit manipulation in C

IRQ0 (Bin)	10010110	11101011	10010110	11101011
A (Bin)	00100000 Bit 5 high	00100000 Bit 5 high	00001000 Bit 3 high	00001000 Bit 3 high
A (Hex)	0x20	0x20	0x08	0x08
B (Bin)	11011111 Bit 5 low	11011111 Bit 5 low	11110111 Bit 3 low	11110111 Bit 3 low
B (Hex)	0xDF	0xDF	0xF7	0xF7
IRQ & A	00000000 Bit 5 of IRQ0	00100000 Bit 5 of IRQ0	00000000 Bit 3 of IRQ	00001000 Bit 3 of IRQ
IRQ | A	10110110 Bit 5 high	11101011 Bit 5 high	10011110 Bit 3 high	11101011 Bit 3 high
IRQ & B	10010110 Bit 5 low	11001011 Bit 5 low	10010110 Bit 3 low	11100011 Bit 3 low
IRQ | B	11011111	11111111	11110111	11111111

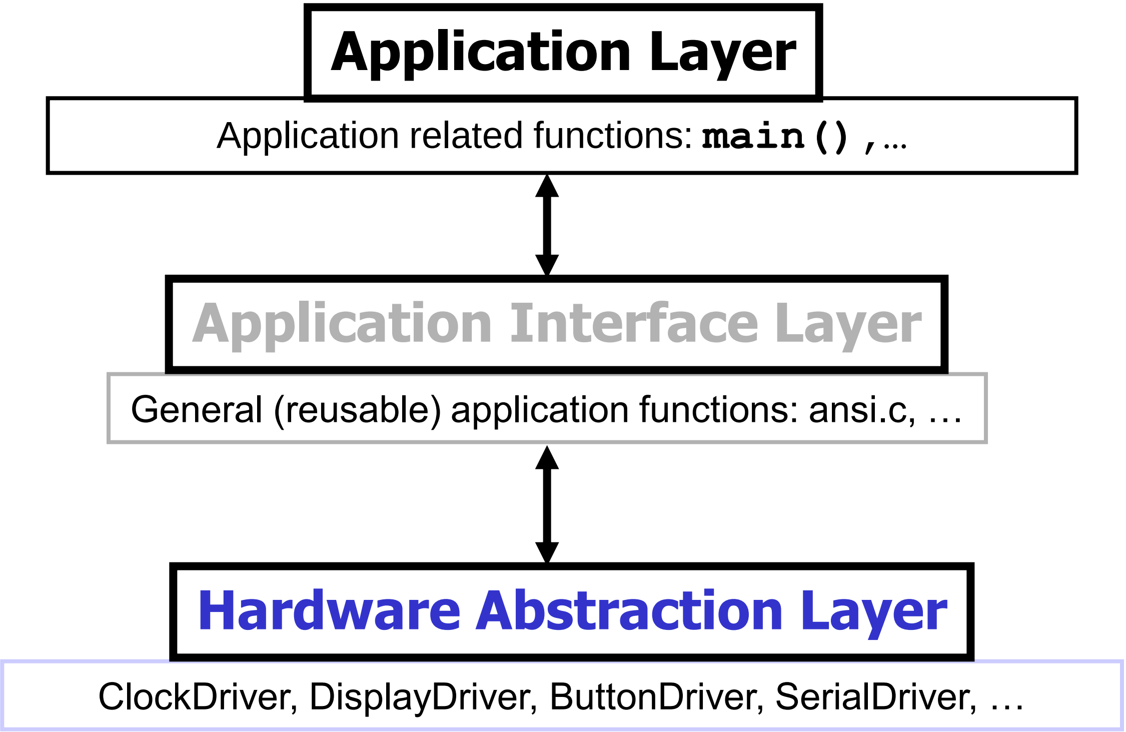
```
IRQ |= 0x20          // sets bit 5 of IRQ
IRQ &= 0xDF          // resets bit 5 of IRQ
(IRQ & 0x20) != 0    // check if bit 5 is set
```

Programming Architecture Abstraction Model


Application Code
Application Programming Interface (API)
RTOS
Middleware
Hardware Abstraction Layer (HAL)
(Drivers and Board Support)
Hardware

Source: https://www.beningo.com/embedded-basics-apis-vs-hals/

Software Architecture



Application Layer

Application related functions: `main(),…`

Application Interface Layer

General (reusable) application functions: ansi.c, …

Hardware Abstraction Layer

ClockDriver, DisplayDriver, ButtonDriver, SerialDriver, …

Application Layer: This is the part of the program that contains the high level specifications. In other words, it is independent of platform and hardware.

Application Interface Layer: This part contains a library with general functions. They are platform independent and core. These can be some basic/standard functions as well as some more specific.

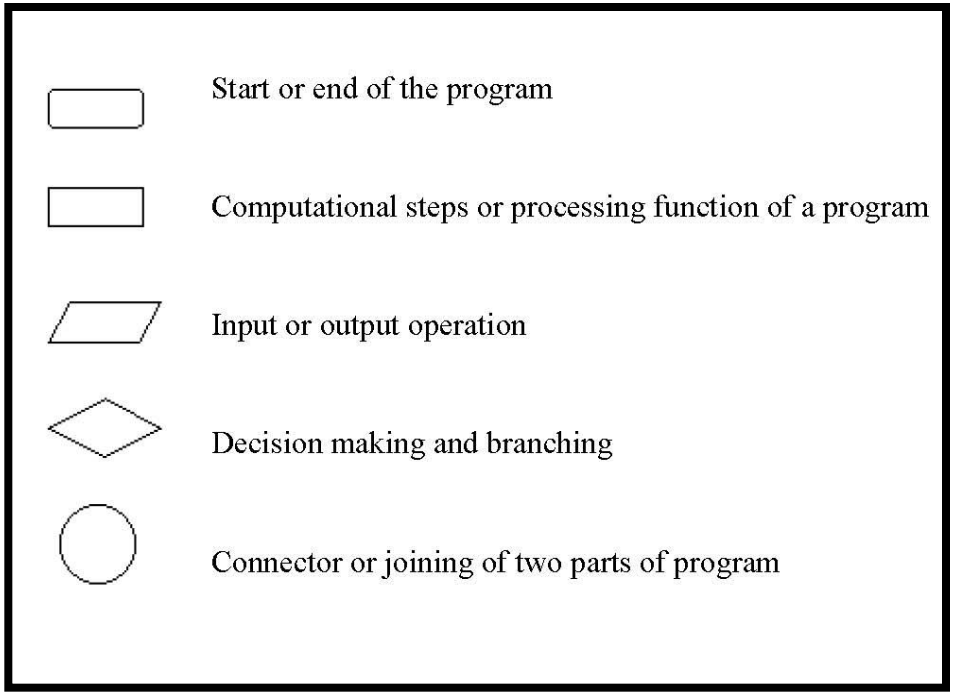
Hardware Abstraction Layer: It is the collection of functions that are strongly hardware dependent. *This is the solely layer that is allowed to ”touch” the hardware*. The functions here give the possibility of interfacing between the hardware and the Application Interface Layer.

This is one of the reasons why:
It is FORBIDDEN to use *Global variables* in this Course

Software Documentation – Application layer

- Flowchart for `main()`

- Symbols for Flowcharting:



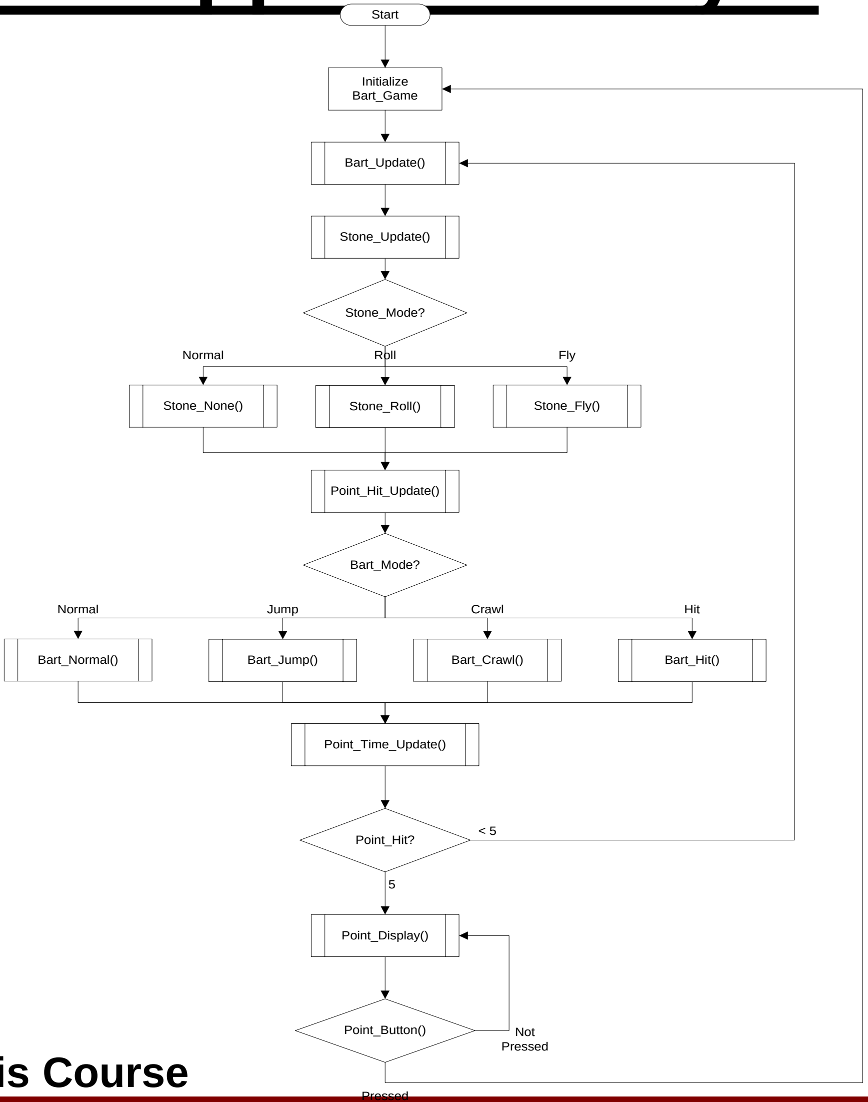
Start or end of the program

Computational steps or processing function of a program

Input or output operation

Decision making and branching

Connector or joining of two parts of program



This is another reason why:
It is FORBIDDEN to use *Global variables* in this Course

Software Documentation – API & HW

Application Interface Layer

Bart.c

Public (Bart.H)

Global

char Bart_Mode

Functions

void Bart_update()[1,2,3]

void Bart_Normal()[1,3]

void Bart_Jump()[1,3]

void Bart_Crawl()[1,3]

void Bart_Hit[1,3]

1 Use Display_Bart in HAL Display.H
2 Use the buttons PD7 and PD6 of Button_Read() in HAL Button.H
3 Use virtual timer 0 in HAL Clock.H

void Bart_Jump()

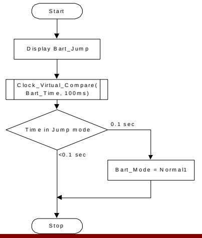
Description:

Displays Bart in lifted position in 1 second. After completion the Bart_Mode is returned to normal mode (0).

Resources:

Uses a virtual timer (0) of Clock.H to keep track of time.

Flowchart:



Hardware Abstraction Layer

Display.C

Public (Display.H)
Constant
char Display_Sprite_Array[50][7]
Public (Display.H)
Functions
void Display_Init()[1]
void Display_Update()[1]
void Display_Character(char, char, char)[1]
void Display_Auto_Init()[1,2]
Privat
Global
char Display_Matrix[4][7]
char Display_Row_Nr
char Display_Dis_Nr

[1] Use port D and E for the four LED displays
[2] Use Timer 2 and the corresponding interrupt for automatic display update

Function DISPLAY_CHARACTER

Syntax

```
void Display_Character(char disp, char ch, char mode);
```

Description

This HAL function puts a symbol of a character into the `Display_ Matrix` to be displayed on the next Display_Update. The character is put on the display number `disp` and the symbol of the character is determined by the ASCII value of `ch`. The character can be put either as a new character (`mode = 0`) or it can be put on top of the symbol already on the display (`mode = 1`). The custom designed sprites can be addressed using the ASCII values 0-9.

Resources

Use port D and port E.

Updates the private global Display_Matrix with the new symbol.

Parameters

char disp:	Display number (1 – 4) where the character should be written.
char ch	ASCII value of the symbol to be displayed.
char mode	0 – New character; 1 – character on top of the existing symbol.

Return Values

None

Example

```
        #include <ez8.h>
#inclu
```

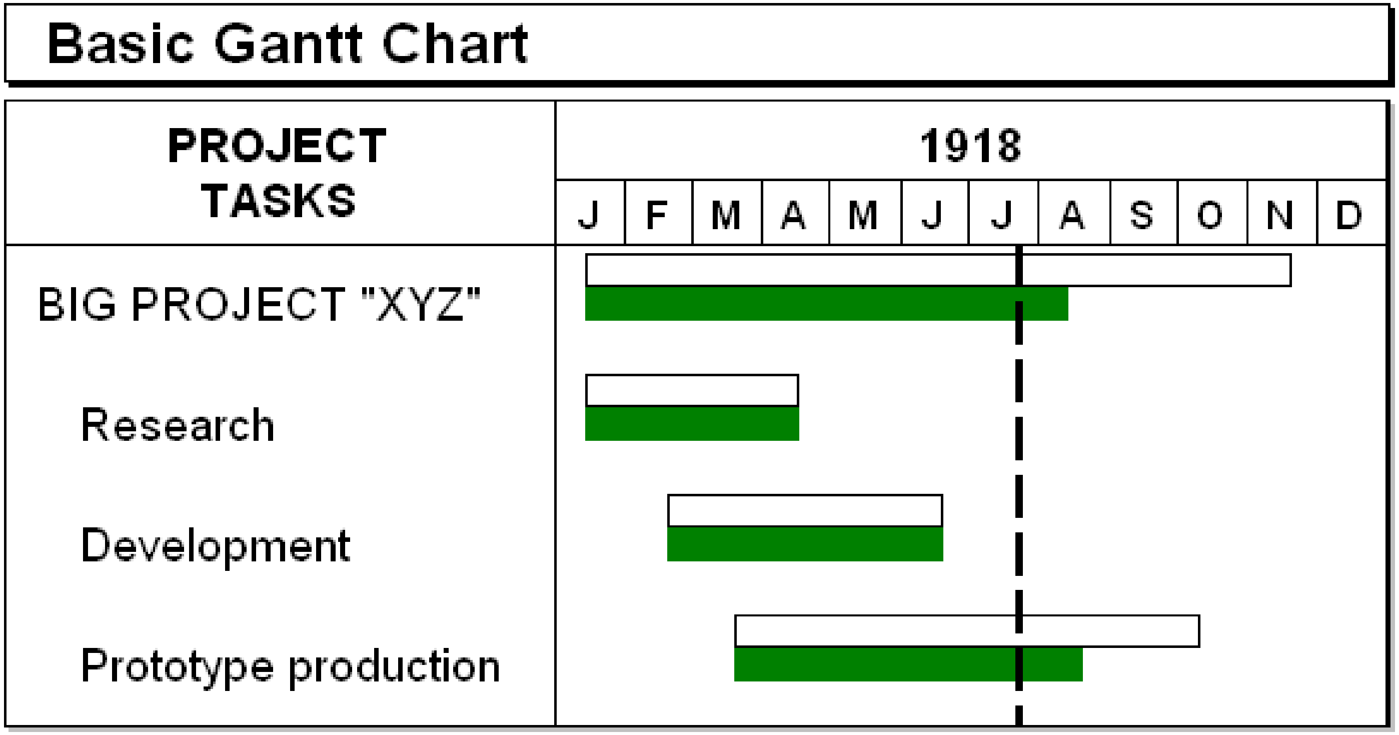
Project Schedule

- A **Gantt chart** is a graphical representation of the duration of tasks against the progression of time. It is an useful tools for planning and scheduling projects:

  - They allow you to assess how long a project should take.

  - Gantt charts lay out the order in which tasks need to be carried out.

(Source: http://www.ganttchart.com/) (FreeSoftware:http://www.smartdraw.com)



Block Diagram

- It is a pictorial representation of some process or model of a complex system. geometric shapes are connected by lines to indicate association and direction/order of traversal.

- It provides a high-level view and is used to :

1) Establish the boundaries of a system under consideration,

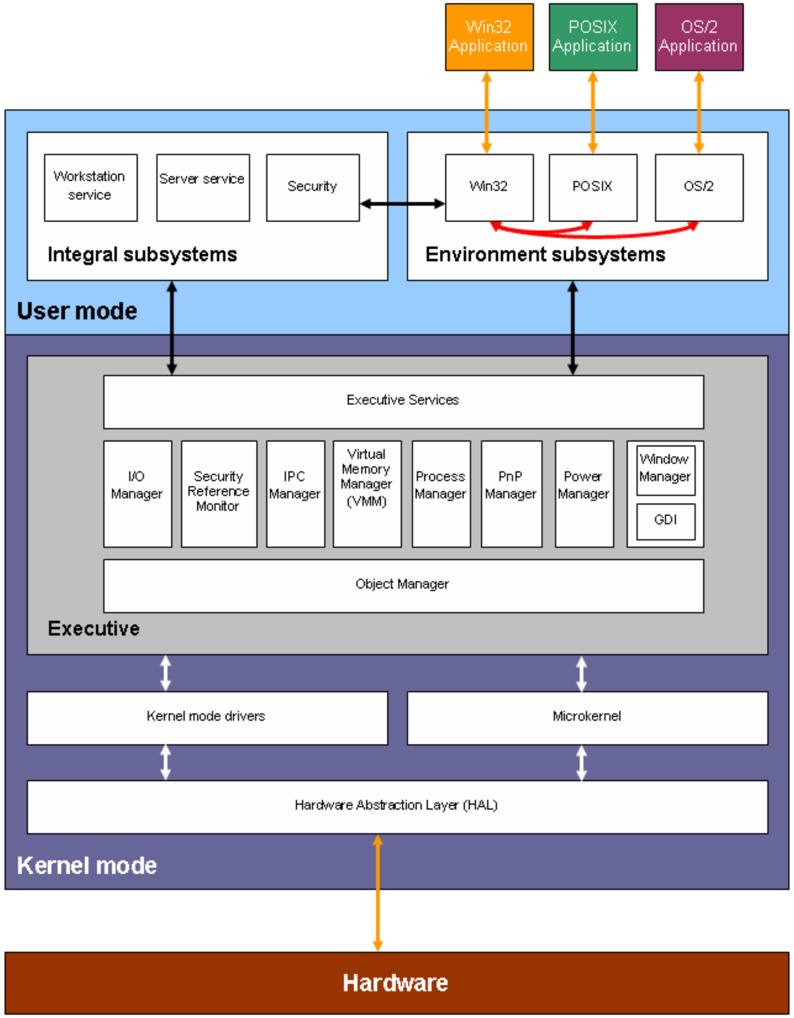
2) Outline the elements contained within the scope of a task - helps Flow Chart

3) Identify inputs and outputs for components within a system,

4) Identify relationships between systems/components,

5) Identify redundancies in systems,

6)Establish critical paths through systems



Example: Windows 2K Architecture

(Source: http://en.wikipedia.org/wiki/Block_diagram
http://thequalityportal.com/q_block.htm)

Tables & Arrays & Pointers

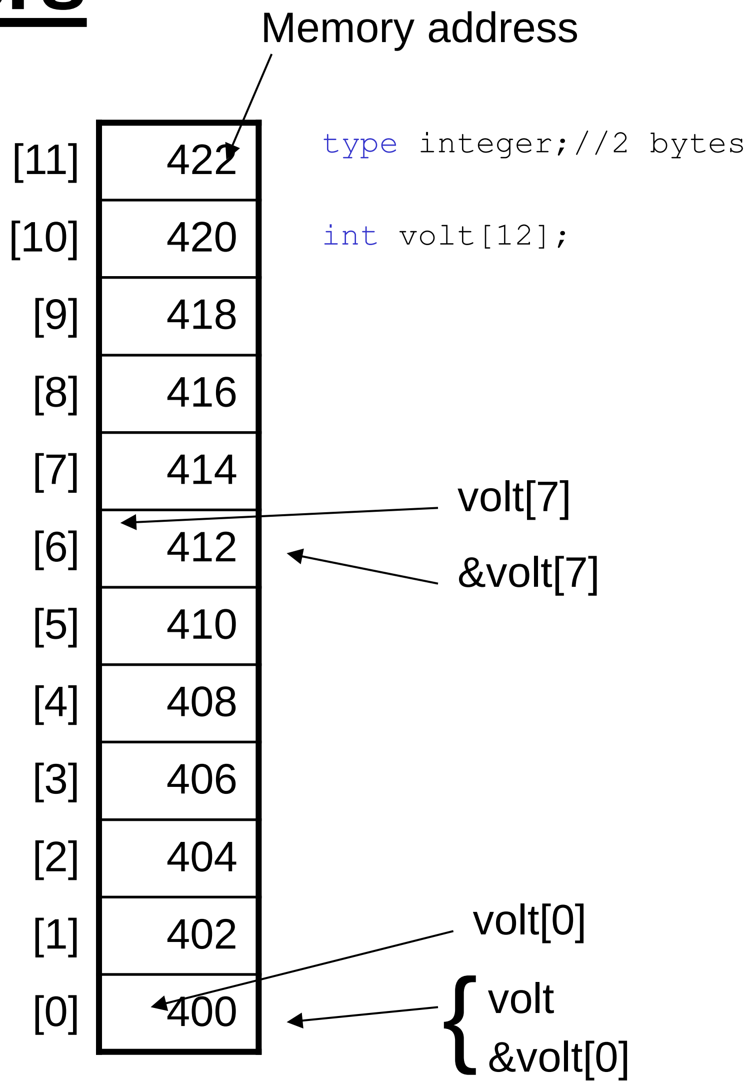
```
#define MAX 5

void main()
   {
    int i,j, max=0,tabel[5];
    int data[MAX]={1,2,3,4,5};
    float tal[]={1.0,2.0,3.0};
    int a[2][3]={ {1,2,3},
                  {4,5,6} };

    for (i=0; i<MAX; i++) tabel[i]=data[i];

    tabel[0]+=5;

    i=1;
    tabel[i++]--;
    tabel[i]=data[MAX]/2+data[0];

    for (i=0; i<2; i++)
       {
        for (j=0; i<3; j++)
           max=max>a[i][j] ? max : a[i][j];
       }

   }
```

```
int a[5];    // a er en tabel med 5 heltal

int *a[5];   // a er en tabel med 5
pointere til heltal

int (*a)[5]; // a er en pointer til en
tabel med 5 heltal
```

Arrays & Pointers

Pointer: is a variable, which value is an address of a data element of a certain type

```
void main()
   {
    int i,j;
    int *ptr;      // ptr er en pointer til en int

    i=7;

    ptr=&i;        // & er en addresse operator
    j=*ptr;        // det samme som i=j=7


    j=*ptr+6;      // j=7+6=13
    j=(*ptr)*10+3; // j=7*10+3=73
    *ptr=-5;       // i=-5

   }


--------------

char *ptr;      // ptr er en pointer til en char
float *ptr;     // ptr er en pointer til en float
```



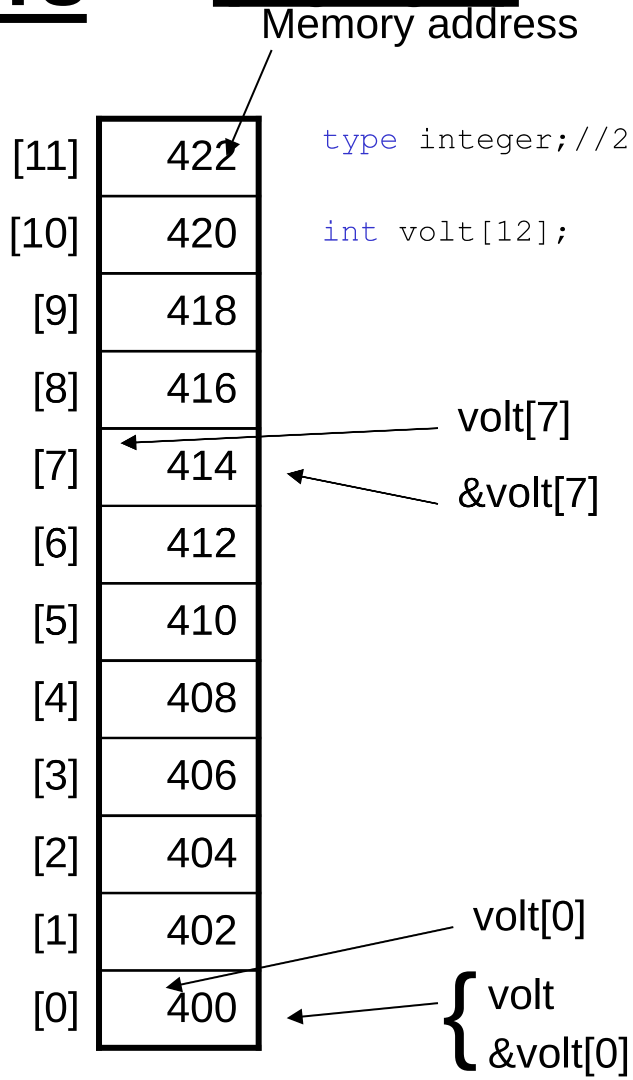
```
type integer;//2 bytes

int volt[12];
```

Arrays & Pointers (rigtigt!)

Pointer: is a variable, which value is an address of a data element of a certain type

```
void main()
   {
    int i,j;
    int *ptr;       // ptr er en pointer til en int

    i=7;

    ptr=&i;         // & er en addresse operator
    j=*ptr;         // det samme som i=j=7


    j=*ptr+6;       // j=7+6=13
    j=(*ptr)*10+3;  // j=7*10+3=73
    *ptr=-5;        // i=-5

   }


--------------

char *ptr;      // ptr er en pointer til en char
float *ptr;     // ptr er en pointer til en float
```

```
type integer;//2 bytes

int volt[12];
```



Memory address

[11]	422
[10]	420
[9]	418
[8]	416
[7]	414
[6]	412
[5]	410
[4]	408
[3]	406
[2]	404
[1]	402
[0]	400

volt[7]

&volt[7]

volt[0]

volt
&volt[0]

Pointer “*Manipulation*”

```
void main()
   {
    // Trin A - Erklæringer

    int a=1,b=2,c=3,d,e;
    int *ptr=&a;

    // Trin B - Her starter selve programmet

    d= *ptr;     // d=a,   ptr=400
    e= *(ptr+2); // e=c,   ptr=400
    *ptr+=5;     // a=a+5, ptr=400

    // Trin C

    ptr++;       //        ptr=402
    d= *ptr++;   // d=b,   ptr=404
    e= *ptr++;   // e=c,   ptr=406

    // Trin D

    (*ptr)++;    // d=d+1, ptr=406

}
```

Adrs.	Var.	A	B	C	D
400	a	1	6	6	6
402	b	2	2	2	2
404	c	3	3	3	3
406	d	-	1	2	3
408	e	-	3	3	3
410	ptr	400	400	406	406
*)	[ptr]	1	6	2	3
*) value of what the pointer is pointing to					

Pointers and Function Arguments

- **Background**:
  - C passes arguments to functions by value, therefore there is not possible for the called function to alter a variable in the calling function.
  - In a sorting routine, we want to exchange 2 numbers with the function `swap`. It is not enough to write:

```
swap (a,b);
```

  - Where the `swap` function is defined as

```
void swap (int x, int y)
  {
   int temp;
   temp=x;
   x=y;
   y=temp;
  }
```

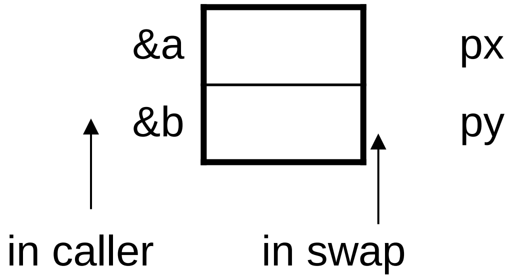
  - The `swap` function can not change the arguments a and b. This function only swaps copies of a and b.

- **Solution**:
  - The way to obtain the desired effect is for the calling program to pass **pointers** to the values to be changed.

```
swap (&a,&b);
```

  - Since the operator & produces the address of a variable, &a is a pointer to a.

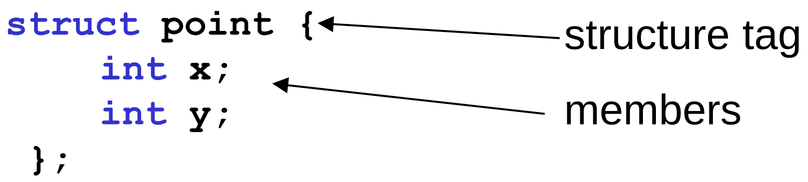
```
void swap (int *px, int *py)
  {
   int temp;
   temp=*px;
   *px=*py;
   *py=temp;
  }
```



Structures

- A structure is a compound data object. A compound data object consists of a collection of data objects of, possibly, different types.

- The graphic basic object is a point of x and y coordinates. A structure for this is:



```
struct point {
    int x;
    int y;
 };
```

structure tag

members

- A `struct` defines a type: `struct { … …} x, y, z;` (like `int x,y,z;`)

- `struct point pt={320,200};` defines and initializes the variable `pt`

- A structure member is referred in expressions as: `structure-name.member`

- The coordinates of a point `pt` are, for instance: `pt.x` and `pt.y`

Structures And Pointers

```
struct TVector {
  char x;
  char y;
};
```

Without pointers:

```
void initVector(struct TVector v) {
  v.x = 10;
  v.y = 20;
}

...
initVector(vec);
...
```

Memory:

Address:	Value:	Variable name:
....		
1000h	?	vec.x
1001h	?	vec.y
1002h	10	copy of vec.x
1003h	20	copy of vec.y
1004h	?	-
1005h	?	-
....		

With pointers:

```
void initVector(struct TVector *v) {
  (*v).x = 10; // or v->x = 10;
  (*v).y = 20; // or v->y = 20;
}

...
initVector(&vec);
...
```

Memory:

Address:	Value:	Variable name:
....		
1000h	10	vec.x
1001h	20	vec.y
1002h	1000h	&vec (pointer to vec)
1003h	1001h	-
1004h	?	-
1005h	?	-
....		

Typical errors in C

General:

- Forgetting to put a break in a `switch` statement
- Using = instead of ==
- Forgetting to put an ampersand (&) on arguments on certain functions
- Using the wrong format for operand
- Size of arrays (in C start index is 0)
- /: Integer division vs. float division
- Loop errors, fx: `while (udtryk)` “;”
- Forgetting prototypes
- Forgetting to initialize pointers
- Interchange ++n and n++. Block { }

String:

- Confusing character and string constants
- Comparing strings with ==
- Not nul terminating strings

Input/Output:

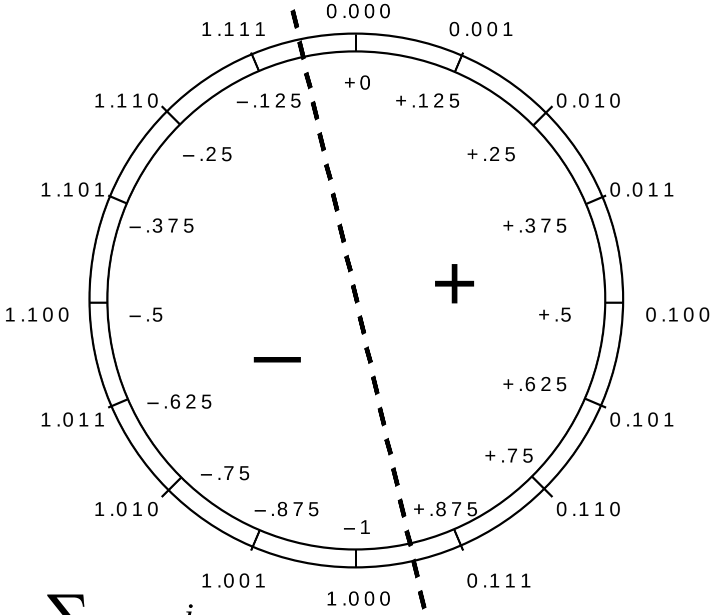
- Using `fgetc()`, etc. incorrectly
- Using `feof()` incorrectly
- Leaving characters in the input buffer
- Using the `gets()` function

The use of *Global variables* generate programming errors, which are very difficult to debug

Comments/Feedback to Exercise 1-2 (1st day)

- gotoxy(): "Moves cursor to line#, column#“
  - ->ANSI Escape codes uses y,x instead of x,y with gotoxy();
- What does init_usb_uart(..)?
  - -> Necessary for writing to Putty
- Why should be a infinite loop at the end of main()?
  - -> It is on the datasheet!
  - -> Because a uP does not return to any place in an OS
  - -> And ST/ARM has not implement it
  - How many did forget this infinite loop?
- Variable shall be defined (initialized also if needed) in each function!
- Check COMx ports (it can be other than COM3-4)
- ANSI: American National Standards Institute
- ASCII - American Standard Code for Information Interchange (complete ASCII table (hex+dec) at: http://www.lookuptables.com/)

- **Exercise 3:**



– Fixed Point Arithmetic

$$0.375 = (0.011)_2 = (0\times2^{0}) + (0\times2^{-1}) + (1\times2^{-2}) + (1\times2^{-3})$$

$$x = (x_{k-1}x_{k-2}\ldots x_1x_0.x_{-1}x_{-2}\ldots x_{-l})_r = \sum x_i\, r^{\,i}$$

– Macro:

```
#define FIX14_SHIFT 14
#define FIX14_MULT(a, b)   ( (a)*(b) >> FIX14_SHIFT )
```

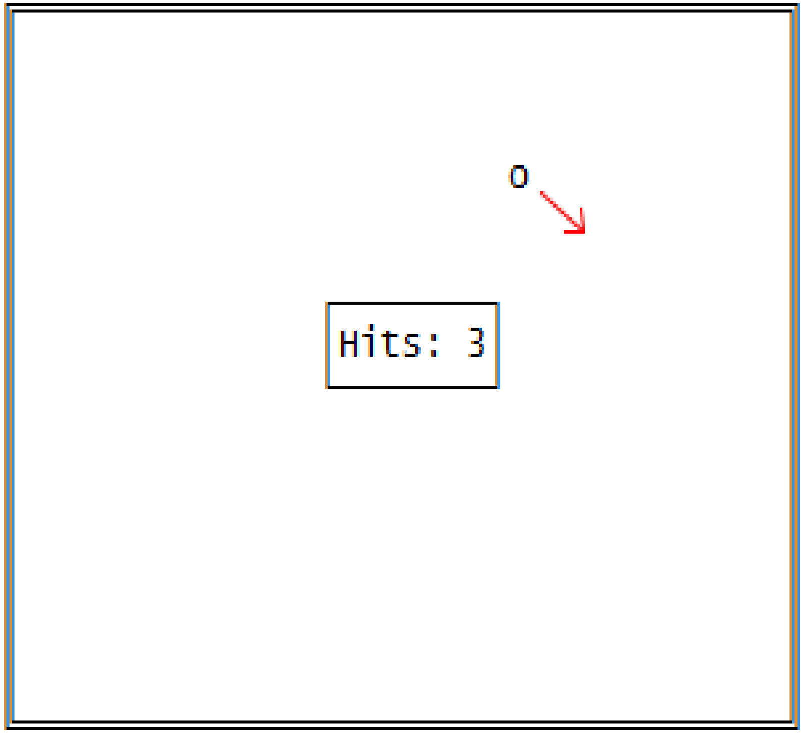
– Create a LookUpTable

– Develop Sin & Cos Functions using their periodic properties

– Rotate Vectors (using structures)

– Bit Manipulation (Individual)

- **<u>Exercise 4</u>:**
  - Ball bouncing between walls
  - Use an structure for position and velocity
  - Detect collisions and reflect
  - Documentation
    - Flowchart
    - Function description
    - Block diagram



```
                 o

            Hits: 3
```