# PROGRAMMAZIONE PROCEDURALE

A.A. 2023/2024

# TYPES

# TYPES

- Programs have to store and process different kinds of data, such as integers and floating-point numbers, in different ways. To this end, the compiler needs to know what kind of data a given value represents.
- In C, the term ***object*** refers to a location in memory whose contents can represent values. Objects that have names are also called ***variables***.
- An object's ***type*** determines:
  - ✓How much space the object occupies in memory.
  - ✓The values that a variable can have.
  - ✓The operations that can be performed on that variable.

# EXAMPLE

```
int main()
{
  int x = 1, y = 2, z = 3;
  printf(" x = %d, y = %d, z = %d \n", x, y, z);
  {
       int x = 10;
       float y = 20;
       printf(" x = %d, y = %f, z = %d \n", x, y, z);
       {
             int z = 100;
             printf(" x = %d, y = %f, z = %d \n", x, y, z);
       }
  }
  return 0;
}
```

# TYPES IN C

- **Basic** type
  - ✓Standard and extended integer types
  - ✓Real and complex floating-point types
- Enumerated types
- The type void
- **Derived** types
  - ✓Pointer types
  - ✓Array types
  - ✓Structure types
  - ✓Union types
  - ✓Function types

# TYPES

- The basic types and the enumerated types together make up the ***arithmetic types***.
- The **arithmetic types and the pointer types** together are called the ***scalar types***.
- Finally, **array types** and **structure types** are referred to collectively as the ***aggregate types***.
- A ***function type*** describes the interface to a function; that is, it specifies the type of the function’s return value, and may also specify the types of all the parameters that are passed to the function when it is called.

# INTEGER

# INTEGERS

- There are five signed integer types. Most of these types can be designated by several synonyms
  - ✓signed char
  - ✓int signed, signed int
  - ✓short short int, signed short, signed short int
  - ✓long long int, signed long, signed long int
  - ✓long long *(C99)* long long int, signed long long, signed long long int
- C defines only the *minimum* storage sizes of the other standard types: the size of type short is at least two bytes, long at least four bytes, and long long at least eight bytes. Furthermore, although the integer types may be larger than their minimum sizes, the sizes implemented must be in the order:
  - ✓sizeof(short) ≤ sizeof(int) ≤ sizeof(long) ≤ sizeof(long long)

# BOOLEANS

- In C there is no true or false
- 0 is false
- Any value different from 0 is true
  - ✓ 1
  - ✓ -25
  - ✓ 123456

```
if (3)
    printf(”YES\n");
else
    printf("NO\n");
```

```
if (0)
    printf(”YES\n");
else
    printf("NO\n");
```

# BOOLEANS

- C99 introduced the unsigned integer type _Bool to represent Boolean truth values.
- The Boolean value *true* is coded as 1, and *false* is coded as 0.
- If you include the header file *stdbool.h* in a program, you can also use the identifiers *bool*, *true*, and *false*. The macro *bool* is a synonym for the type *_Bool*, and *true* and *false* are symbolic constants equal to 1 and 0.

# CHARS

- The type char is also one of the standard **integer types**. However, the one-word type name char is synonymous either with signed char or with unsigned char, depending on the compiler.
- It occupies 1 byte
- Check the correspondence in the ASCII table
  - ✓http://www.asciitable.com

# CHARS

- You can do arithmetic with character variables. It’s up to you to decide whether your program interprets the number in a char variable as a character code or as something else.

```
char ch = 'A';                // A variable with type char.
printf("The character %c has the character code %d.\n", ch, ch);
printf(”%c", ch + 1);
```

```
The character A has the character code 65.
B
```

# BINARY/OCTAL/HEXADECIMAL

# BINARY NUMERAL SYSTEM

- In mathematics and digital electronics, a binary number is a number expressed in the binary numeral system or base-2 numeral system which represents numeric values using two different symbols: typically 0 (zero) and 1 (one).
- Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used internally by almost all modern computers and computer-based devices. Each digit is referred to as a bit.

1000101

# HEXADECIMAL

- Hexadecimal (also base 16, or hex) is a positional numeral system with a radix, or base, of 16.
- It uses sixteen distinct symbols, most often the symbols 0–9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen.
- Hexadecimal numerals are widely used by computer system designers and programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values.
- One hexadecimal digit represents a nibble (4 bits), which is half of an octet or byte (8 bits).

f1a2

# OCTAL

- The **octal** numeral system, is the base-8 number system, and uses the digits 0 to 7.

# FROM BINARY AND HEXADECIMAL TO DECIMAL

$$d_n 2^n + d_{n-1} 2^{n-1} + \ldots + d_1 2^1 + d_0 2^0 = N_{10}$$

1001 = 9

10~~2~~1 = ?

002B = 43

| Binary | Hexadecimal | Decimal |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 10 | 2 | 2 |
| 11 | 3 | 3 |
| 100 | 4 | 4 |
| 101 | 5 | 5 |
| 110 | 6 | 6 |
| 111 | 7 | 7 |
| 1000 | 8 | 8 |
| 1001 | 9 | 9 |
| 1010 | A | 10 |
| 1011 | B | 11 |
| 1100 | C | 12 |
| 1101 | D | 13 |
| 1110 | E | 14 |
| 1111 | F | 15 |

# FROM DECIMAL TO BINARY

| ÷2 | remainder |
|---|---|
| 156 | 0 |
| 78 | 0 |
| 39 | 1 |
| 19 | 1 |
| 9 | 1 |
| 4 | 0 |
| 2 | 0 |
| 1 | 1 |
| 0 | |

10011100 = 156

# FROM DECIMAL TO HEXADECIMAL

| | ÷16 | remainder |
|---|---|---|
| | 1565 | 13 = d |
| | 97 | 1 |
| | 6 | 6 |
| | 0 | |

61d= 1565

Octal follows the same algorithm

# HOW MANY NUMBERS CAN I REPRESENT?

- From 0 to $(2^N - 1)$
- With 8 bits (one byte)
  - ✓From 0 to 255

    0000 0000 1111 1111

- However, it is useful to also represent negative numbers
- Different representations
  - ✓Sign and magnitude
  - ✓Two's complement

# REPRESENTATION IN MEMORY

# SIGN AND MAGNITUDE

- It uses one bit (usually the leftmost if big endian) to indicate the sign. "0" indicates a positive integer, and "1" indicates a negative integer. The rest of the bits are used for the magnitude of the number. E.g.:
  - ✓1001 1000
  - ✓-24
- if 1001 1000 is used to represent positive numbers only?
  - ✓152

# HOW MANY NUMBERS CAN I REPRESENT?

- With n bits
  - ✓From $(-2^{N-1} + 1)$ to $(2^{N-1} - 1)$
  - ✓ and ±0

- For instance, with 8 bits,
  - ✓from -127 to + 127

1111 1111        0111 1111

# A PROBLEM

- Two different representations of 0
  - ✓0000 0000 (+0)
  - ✓1000 0000 (-0)

- A solution is a different representation: two's complement

# TWO’S COMPLEMENT

| Binary value | Two's complement | Unsigned |
|---|---|---|
| 00000000 | 0 | 0 |
| 00000001 | 1 | 1 |
| ⋮ | ⋮ | ⋮ |
| 01111110 | 126 | 126 |
| 01111111 | 127 | 127 |
| 10000000 | −128 | 128 |
| 10000001 | −127 | 129 |
| 10000010 | −126 | 130 |
| ⋮ | ⋮ | ⋮ |
| 11111110 | −2 | 254 |
| 11111111 | −1 | 255 |

In two's-complement, there is only one zero, represented as 00000000. Negating a number (whether negative or positive) is done by inverting all the bits and then adding one to that result

# HOW TO GET THE COMPLEMENTARY

- From any number to its complement:
  - ✓Flip all the bits (0->1, 1->0) and then + 1

- 0000 0101 (value 5)
  - ✓1111 1010 (flip)
  - ✓1111 1011 (+1)
- 1111 1011 value (-5)
  - ✓0000 0100 (flip)
  - ✓0000 0101 (+1)

# HOW MANY NUMBERS CAN I REPRESENT?

- With n bits
  - ✓From $(-2^{N-1})$ to $(2^{N-1} - 1)$
  - ✓ There is no "-0", so it is possible to represent one more negative number

- For instance, with 8 bits,
  - ✓from -128 to + 127

    1000 0000 0111 1111

- The rule in the previous slide to get the complimentary does not work because 128 is not representable with 8 bits in two's complement

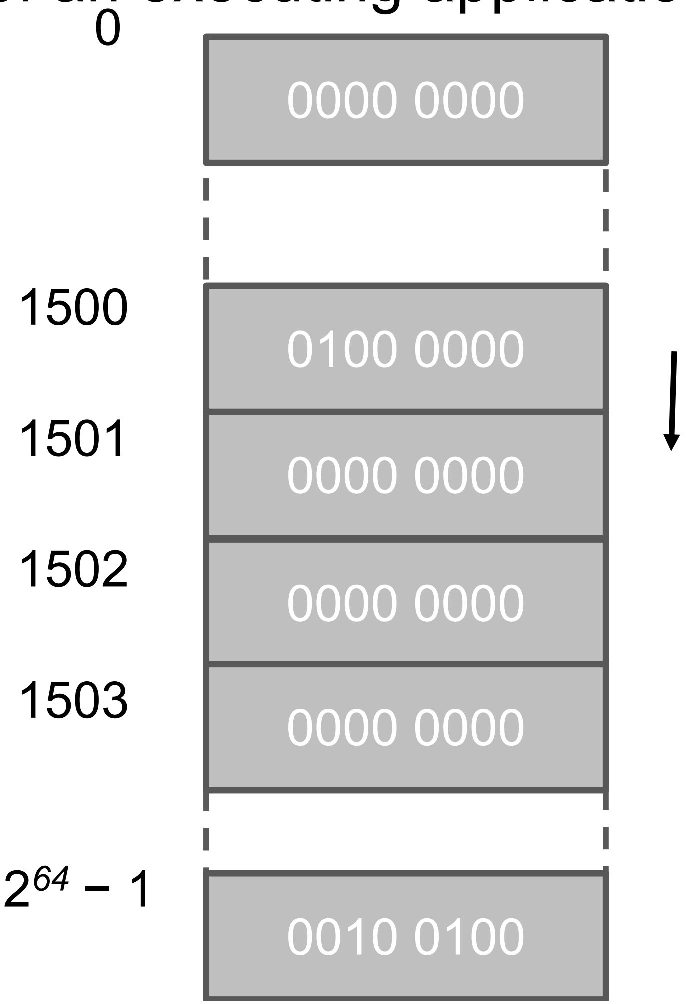
# ENDIANESS

- Endianness refers to the sequential order used to numerically interpret a range of bytes in computer memory as a larger, composed word value.
- It also describes the order of byte transmission over a digital link.
- Words may be represented in **big-endian** or **little-endian** format, depending on whether bits or bytes or other components are numbered from the big end (most significant bit) or the little end (least significant bit).
- As examples, the IBM z/Architecture mainframes and the Motorola 68000 series use big-endian while the Intel x86 processors use little-endian (in the 1970s).

# MEMORY REPRESENTATION

- An array of bytes
- Every byte has its logical address (a positive number)
- A **logical address** is the **address** at which an item appears to reside from the perspective of an executing application program.

2 represented on 4 bytes (little end.):
0100 0000
0000 0000
0000 0000
0000 0000




For 64-bit architectures the upper limit $2^{64} - 1$

# BIG ENDIAN, LITTLE ENDIAN

Big endian: right to left

Little endian: left to right

Two's complement from now on

Big endian:
0010=
2

Little endian
0010 =
4

Big endian:
1010=
-6

Little endian
1010 =
5

Big endian


Little endian

BACK TO C

# REPRESENTATION

| Type | Storage size | Minimum value | Maximum value |
|---|---|---|---|
| char | (same as either signed char or unsigned char) | | |
| unsigned char | one byte | 0 | 255 |
| signed char | one byte | −128 | 127 |
| int | two bytes or four bytes | −32,768 or -2,147,483,648 | 32,767 or 2,147,483,647 |
| unsigned int | two bytes or four bytes | 0 | 65,535 or 4,294,967,295 |
| short | two bytes | −32,768 | 32,767 |
| unsigned short | two bytes | 0 | 65,535 |
| long | ~~four bytes~~ 8 bytes | −2,147,483,648 | 2,147,483,647 |
| unsigned long | ~~four bytes~~ | 0 | 4,294,967,295 |
| long long *(C99)* | eight bytes | −9,223,372,036, 854,775,808 | 9,223,372,036, 854,775,807 |
| unsigned long long *(C99)* | eight bytes | 0 | 18,446,744,073, 709,551,615 |

So, how are integer represented in C? Sign magnitude or two's complement?

# SIZEOF

- To obtain the exact size of a type or a variable, use the *sizeof* operator. The expressions *sizeof(type)* and *sizeof expression* yield the storage size of the object or type in bytes. If the operand is an expression, the size is that of the expression's type.

```
int iIndex,
iIndex = 1000;
```

- sizeof(int) and sizeof(iIndex) returns 4

# LIMITS

- You can find the value ranges of the integer types for your C compiler in the header file *limits.h*, which defines macros such as INT_MIN, INT_MAX, UINT_MAX, and so on

```
int main() {
    printf(" char %d %d %d\n", sizeof(char), CHAR_MIN, CHAR_MAX );
    printf(" int %d %d %d\n", sizeof(int), INT_MIN, INT_MAX );
    return 0;
}
```

FLOAT

# FLOATING POINT TYPES

- C also includes special numeric types that can represent non-integers with a decimal point in any position. The standard *floating-point types* for calculations with real numbers are as follows:
  - ✓float:  for variables with single precision
  - ✓double: for variables with double precision
  - ✓long double: for variables with extended precision

# PRECISION

- A floating-point value can be stored only with a limited precision, which is determined by the binary format used to represent it and the amount of memory used to store it.
- The precision is expressed as a number of significant digits. So that its conversion back into a six-digit decimal number yields the original six digits.
- The position of the decimal point does not matter, and leading and trailing zeros are not counted in the six digits.
  - ✓The numbers 123,456,000 and 0.00123456 can both be stored in a type with six-digit precision.

# AR. OPERATIONS IN DOUBLE PREC.

- In C, arithmetic operations with floating-point numbers are performed internally with double or greater precision.

```
float height = 1.2345, width = 2.3456;    // Float variables have single
                                          // precision.
double area = height * width;             // The actual calculation is
                                          // performed with double
                                          // (or greater) precision.
```

- If you assign the result to a float variable, the value is rounded as necessary.

# FLOATS

- The header file *float.h* defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. The macros FLT_MIN, FLT_MAX, and FLT_DIG indicate the value range and the precision of the float type.

| Type | Storage size | Value range | Smallest positive value | Precision |
|---|---|---|---|---|
| `float` | 4 bytes | ±3.4E+38 | 1.2E–38 | 6 digits |
| `double` | 8 bytes | ±1.7E+308 | 2.3E–308 | 15 digits |
| `long double` | 10 bytes | ±1.1E+4932 | 3.4E–4932 | 19 digits |

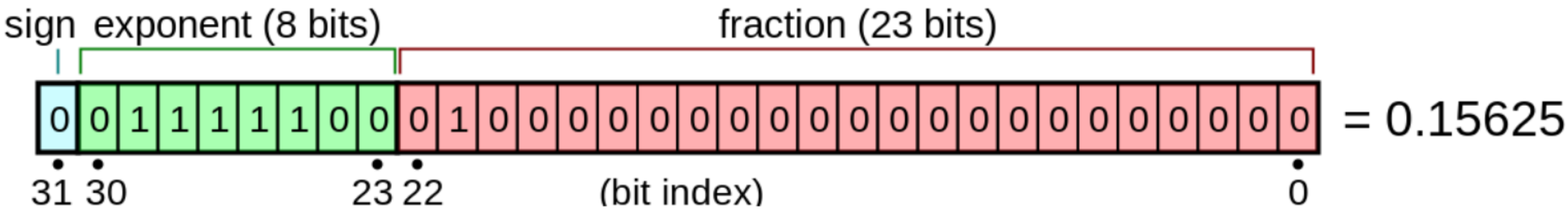
# E NOTATION

- It's know as E notation, which is plain text representation of scientific notation.
  - ✓1.234e+56 means $1.234 \times 10^{56}$

# IEEE 754 FORMAT

- Each finite number is described by three integers: $s$ = a *sign* (zero or one), $c$ = a significand (or "mantissa"), q = an exponent. The numerical value of a finite number is
  - ✓ $(-1)^s \times c \times b^q$
  - ✓ where $b$ is the base (e.g., 2 or 10), also called *radix*.
- For example, if the base is 10, the sign is 1 (indicating negative), the significand is 12345, and the exponent is −3, then the value of the number is −12.345
- The 754 format for single precision is
  - ✓ Sign 1 bit
  - ✓ Exponent 8 bits
  - ✓ Significand 23 bit

# EXAMPLE



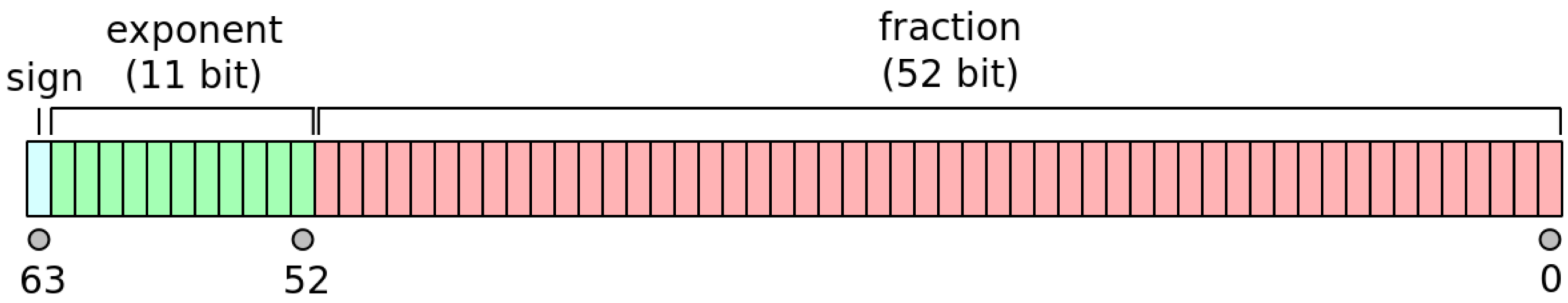

$$\text{value} = (-1)^{\text{sign}} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right) \times 2^{(e-127)}$$

$1 + 1 \times 2^{-2}$ 124 - 127

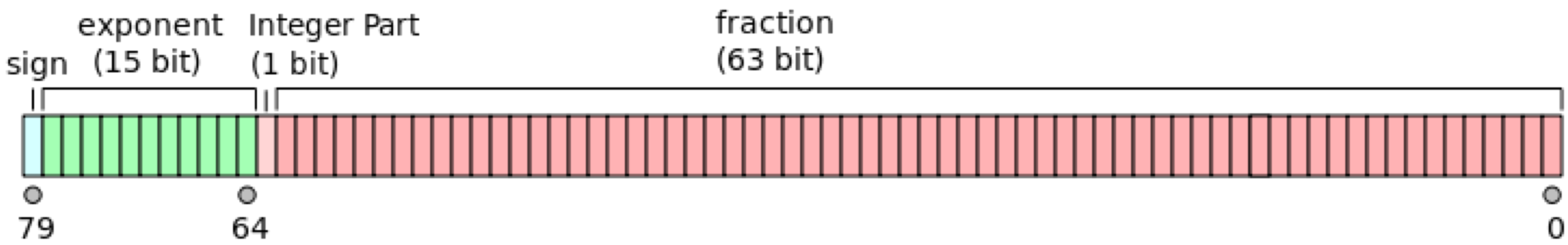
$$\text{value} = (+1) \times 1.25 \times 2^{-3} = +0.15625$$

# DOUBLE AND EXTENDED PRECISION

Double precision: **double** in C




Extended precision: **long double** in C



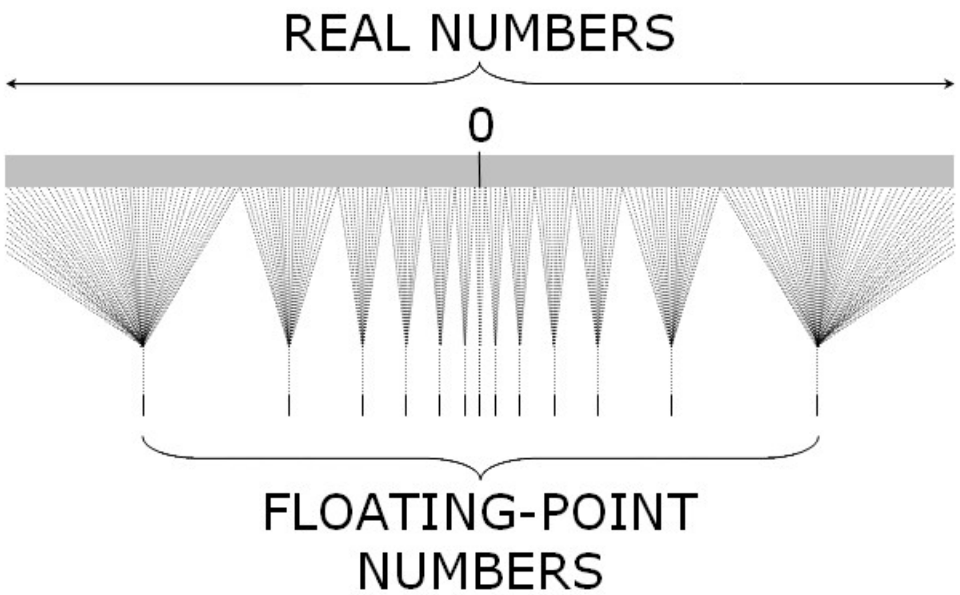
# EXAMPLES

Rounding error is inherent in floating-point computation

```
#include <stdio.h>

int main(){
    int a= 16777217;
    float b= a;
    printf("%f\n", b);
}
```

```
16777216.000000
```



# MORE ERRORS

```
float height = 1.2345, width = 2.3456; // Float variables have
                                        // single precision.

double area = height * width; // The actual calculation
                              // is performed with
                              // double precision.
```

```
float height = 1.2345, width = 2.3456;.

float area = height * width;
```

# HOW TO AVOID PROBLEMS

- The easiest way to avoid accumulating error is to use high-precision floating-point numbers (this means **using double instead of float**). On modern CPUs there is little or no time penalty for doing so, although storing doubles instead of floats will take twice as much space in memory.

# ENUM

# ENUM TYPES

- Enumerations are integer types that you define in a program
- The definition of an enumeration begins with the keyword enum, possibly followed by an identifier for the enumeration, and contains a list of the type's possible values, with a name for each value:

  **enum [*identifier*] { *enumerator-list* };**
- An example is
  - ✓enum color { black, red, green, yellow, blue, white=7, gray };
  - ✓the constants listed have the values 0, 1, 2, 3, 4, 7, 8

# EXAMPLES

```
enum color fgColor = blue,          // Define two variables
bgColor = yellow;                   // of type enum color.

void setFgColor( enum color fgc ); // Declare a function with a parameter
                                    // of type enum color.
```

- You may perform ordinary arithmetic operations with variables of enumerated types:
  - ✓ red + red = 2
- Different constants in an enumeration may have the same value:
  - ✓ enum signals { OFF, ON, STOP = 0, GO = 1, CLOSED = 0, OPEN = 1 };

# MORE EXAMPLES

- enum boolean { false, true };
  - ✓enum boolean check;

```
#include <stdio.h>

enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };

int main(){

    enum week today;
    today = wednesday;
    printf("Day %d",today+1);
    return 0;
}
```

Day 4

# WHEN TO USE ENUM

- You should always use enums when a variable (especially a method parameter) can only take one out of a small set of possible values.

- If you use enums instead of integers, your avoid errors from passing in invalid constants, and you document which values are legal to use. Moreover, it is more mnemonic to use them instead of integer values.

VOID

# VOID

- The type specifier void indicates that **no value is available**.
- Consequently, you cannot declare variables or constants with this type, but you can use
  - ✓void in function declarations
  - ✓Void expressions
  - ✓Pointers to void (<u>back to this when we will study pointers</u>)

# VOID IN FUNCTIONS

- A function with no return value has the type void.
  - ✓void error(int a) {}
- void in the parameter list of a function prototype indicates that the function has no parameters:
  - ✓void printMenu(void) {}
- The compiler issues an error message if you try to use a function call such as printMenu(3).

# SU LIBRO

- Pag 423, 424, 635, 636
- Sezione 7.7 (sizeof)
- Sezione 10.11 (enum)
- Sistema binario
  - ✓ https://www.liceomorgagni.edu.it/sites/www.liceomorgagni.it/files/materiali-docente/simboli_cifre_numeri.pdf
  - ✓ http://infodoc.altervista.org/sistemi-di-numerazione-binario-ottale-esadecimale/#_Toc92978982
  - ✓ https://www.math.unipd.it/~aiolli/corsi/0708/infxbio/lez08.pdf