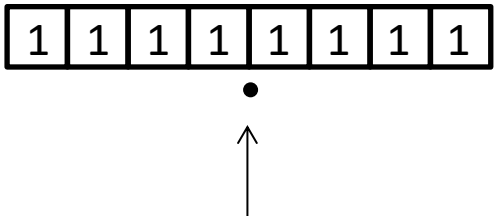
Ceng 111 – Fall 2021
Week 8a

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Binary Representation of Real Numbers

- Approach 1: Use fixed-point
  - Similar to integers, except that there is a decimal point.
  - E.g.: using 8 bits:

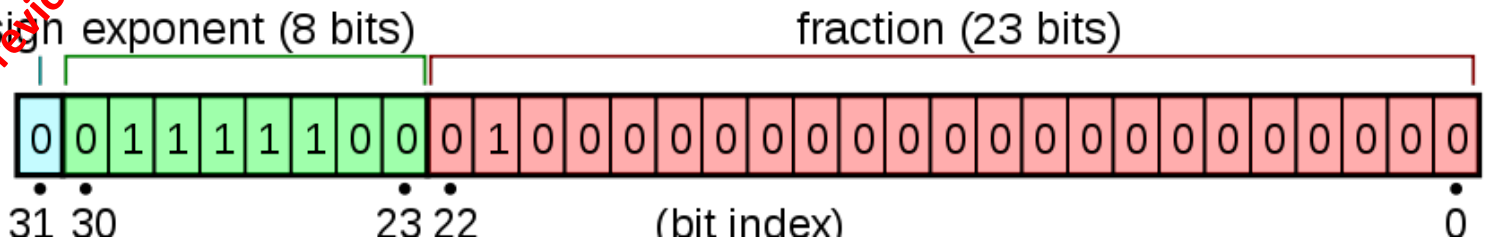


$= 1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}$

$= 15.9375$

IEEE 32bit Floating-Point Number Representation

Previously on CENG111!



$= 0.15625$

$$= (-1)^{\text{sign}}(1.b_{-1}b_{-2}...b_{-23})_2 \times 2^{e-127}$$

- M x 2^E $(2 - 2^{-23}) \times 2^{127}$
- Exponent (E): 8 bits
  - Add 127 to the exponent value before storing it
  - **E can be 0 to 255 with 127 representing the real zero.**
- Fraction (M - Mantissa): 23 bits
- $2^{128} = 1.70141183 \times 10^{38}$

Previously on CENG111!

Why add bias to the exponent?

- It helps in comparing the exponents of the same-sign real-numbers without looking out for the sign of the exponent.

Binary Number	Decimal Value	Value in Two's Complement	Value with bias 7
0000	0	0	-7
0001	1	1	-6
0010	2	2	-5
0011	3	3	-4
0100	4	4	-3
0101	5	5	-2
0110	6	6	-1
0111	7	7	0
1000	8	-8	1
1001	9	-7	2
1010	10	-6	3
1011	11	-5	4
1100	12	-4	5
1101	13	-3	6
1110	14	-2	7
1111	15	-1	8

To read more on this:
https://blog.angularindepth.com/the-mechanics-behind-exponent-bias-in-floating-point-9b3185083528

Previously on CENG111!

IEEE 32bit Floating-Point Number Representation

- Now consider 4.1:
  - 4 => $(100)_2$
  - 0.1 =>
    - x 2 = 0.2 = 0 + 0.2
    - x 2 = 0.4 = 0 + 0.4
    - x 2 = 0.8 = 0 + 0.8
    - x 2 = 1.6 = 1 + 0.6
    - x 2 = 1.2 = 1 + 0.2
    - x 2 = 0.4 = 0 + 0.4
    - x 2 = 0.8 = 0 + 0.8
    - …….

- So,
  - Representing a fraction which is a multiple of $1/2^n$ is lossless.
  - Representing a fraction which is not a multiple of $1/2^n$ leads to accuracy loss.

Previously on CENG 111!

Binary Representation of Textual Information

- Characters are mapped onto binary numbers
  - ASCII (American Standard Code for Information Interchange) code set
    - Originally: 7 bits per character; 128 character codes
  - Unicode code set
    - 16 bits per character
  - UTF-8 (Universal Character Set Transformation Format) code set.
    - Variable number of 8-bits.

How about a text?

- Text in a computer has two alternative representations:

1. A fixed-length number representing the length of the text followed by the binary values of the characters in the text.
   - Ex: “ABC” =>

   00000011 01000001 01000001 01000001 (3 ‘A’ ‘B’ ‘C’)

2. Binary values of the characters in the text ended with a unique marker, like “00000000” which has no value in the ASCII table.
   - Ex: “ABC” =>

   01000001 01000001 01000001 00000000 (‘A’ ‘B’ ‘C’ END)

Basic Data Types

- Integers
  - Full support from the CPU
  - Fast processing
- Floating Points
  - Support from the CPU with some precision loss
  - Slower compared to integer processing due to “interpretation”

Previously on CENG111!

Problems due to precision loss

- 1.0023 – 1.0567
  - Result: -0.05440000000000004
- 1000.0023 – 1000.0567
  - Result: -0.05439999999986903
- Why?
  - Since the floating point representation is based on shifting the bits in the mantissa, the following are not equivalent in a PC
- π = 3.1415926535897931….
  - sin(π) should be zero
  - But it is not: $1.2246467991473532 \times 10^{-16}$

Integers in Python

- Python provides `int` type.

```
>>> type(3)
<type 'int'>
>>> type(3+4)
<type 'int'>
>>>
```

- For big integers, Python has the `long` type:

```
>>> type(3L)
<type 'long'>
>>> type(3L+4L)
<type 'long'>
>>>
```

`int` is limited by the hardware whereas `long` type is unlimited in Python.

Floating Points in Python

- Python provides `float` type.

```
>>> type(3.4)
<type 'float'>
>>>
```

```
>>> 3.4+4.3
7.7
>>> 3.4 / 4.3
0.7906976744186046
```

Previously on CENG111!

Characters in Python

- Python does not have a separate data type for characters!
- However, one character strings can be treated like characters:
  - **ord**(One_Char_String) : returns the ASCII value of the character in One_Char_String.
    - Ex: ord("A") returns 65.
  - **chr**(ASCII_value) : returns the character in a string that has the ASCII value ASCII_value:
    - Ex: chr(66) returns "B"

Previously on CENG111!

Boolean Values in Python

- Python provides the `bool` data type for boolean values.
- `bool` data type can take `True` or `False` as values.

```
>>> 3 > 4
False
>>> type(3 > 4)
<type 'bool'>
```

not operator:

```
not True
not 4 > 3
```

```
>>> True == 2
False
>>> True == 1
True
>>> False == 0
True
>>> False == 1
False
```

Try this:

```
>>> True == bool(2)
```

Today

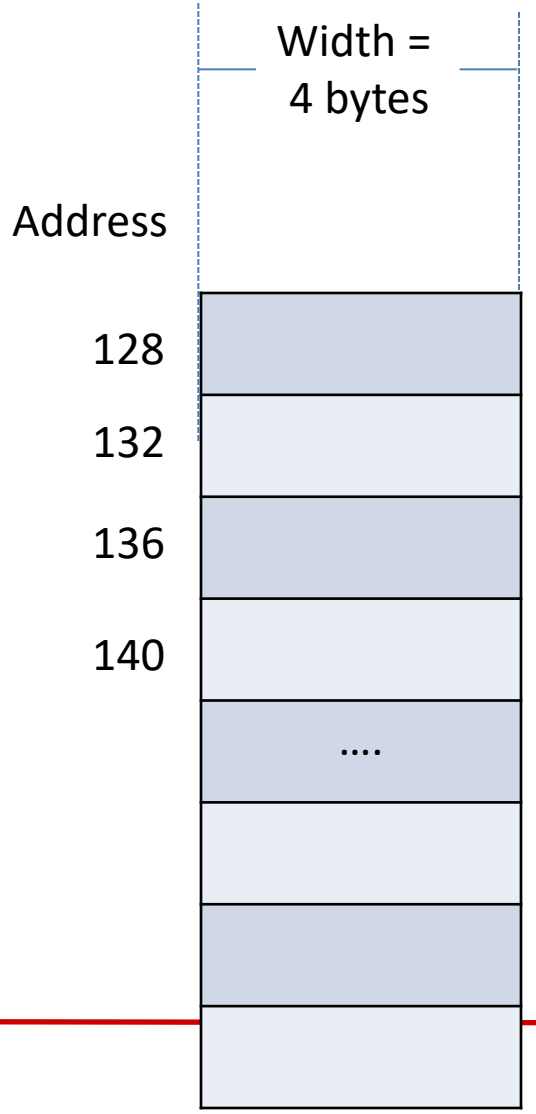
- Container Data

Administrative Notes

- Midterm date:
  - 22 December, Wednesday, 18:00

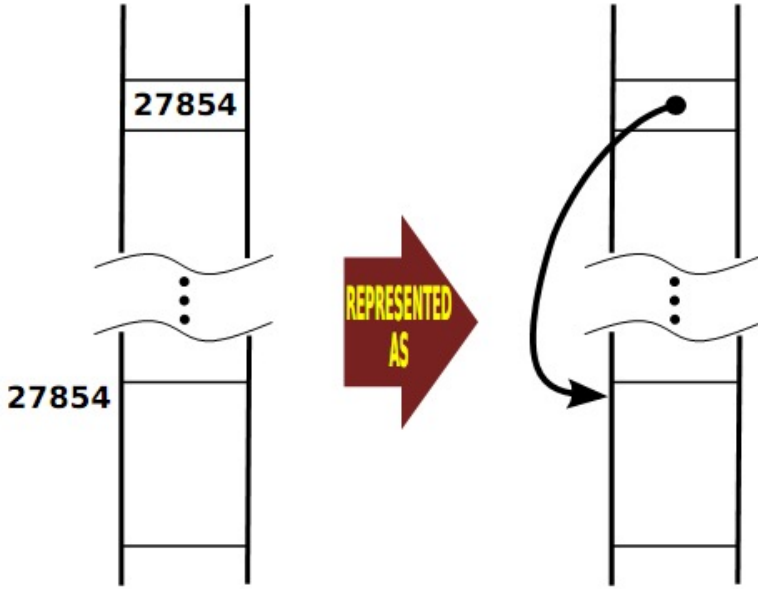
Structured Data

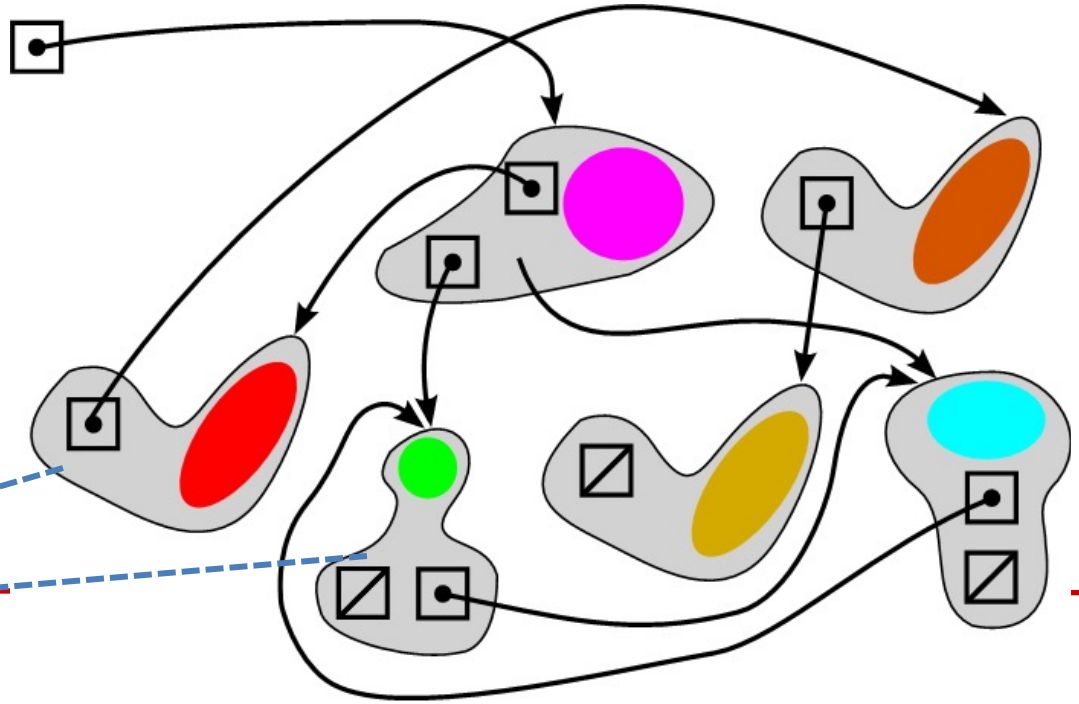
- If you have lots and lots of one type of data (for example, the ages of all the people in Turkey):
  - You can store them into memory consecutively (supported by most PLs)
    - This is called *arrays*.
  - Easy to access an element. Nth element:
    - <Starting-address>+ (N-1)*<Word Width>
    - Ex: 2nd element is at 128 + (2-1) * 4 = 132



Structured Data

- What if you have to make a lot of deletions and insertions in the middle of an array?
- Then, you have to store your data in blocks/units such that each unit has the starting address of the next unit/block.





Strings

- Sequence of characters:
  - Ex: “Book”, “Programming”, “Python”
- How can they be represented?
  1. Put a set of characters one after the other and end them with a non-character value.
  2. At the beginning of the characters, specify how many characters follow.
- Both have advantages and disadvantages.

Strings in Python

Python provides the **str** data type for strings:

```
>>> "Hello?"
'Hello?'
>>> type("Hello?")
<type 'str'>
```

- Simplest operation with a string:

```
>>> len("Hello?")
6
```

Strings in Python

- Accessing elements of a string

  “Hello?”[0] → 1st character (i.e., “H”)

  “Hello?”[4] → 5th character

- Indexing starts at 0!!!
- What is the last element then?
  - “Hello?”[len(“Hello?”) - 1]
- Negative indexing possible:
  - Last element: “Hello?”[-1] → “?”
- In general:
  - String[start:end:step]
  - Ex: “Hello?”[0:4:2] → “Hl”
  - Ex: “Hello?”[2:4] → “ll”

Creating Strings in Python

1. Enclosing a set of characters between quotes:
   - “ali”, “veli”, “deli”, …
2. Using the str() function:
   - str(4.5) → “4.5”
3. Using the raw_input() function:

```
>>> a = raw_input("--> ")
--> Do as I say
>>> a
'Do as I say'
>>> type(a)
<type 'str'>
```

Internal of Python's String Implementation

(taken from https://www.laurentluce.com/posts/python-string-objects-implementation/)

- PyStringObject structure
  - "A string object in Python is represented internally by the structure PyStringObject. "ob_shash" is the hash of the string if calculated. "ob_sval" contains the string of size "ob_size". The string is null terminated. The initial size of "ob_sval" is 1 byte and ob_sval[0] = 0. If you are wondering where "ob_size is defined", take a look at PyObject_VAR_HEAD in object.h."

```
typedef struct {
    PyObject_VAR_HEAD
    long ob_shash;
    int ob_sstate;
    char ob_sval[1];
} PyStringObject;
```