Lecture 01

Introduction to Data Structures

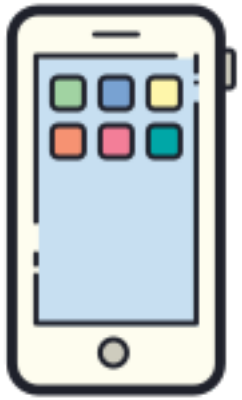
AED-1026: Data Structures, 2019

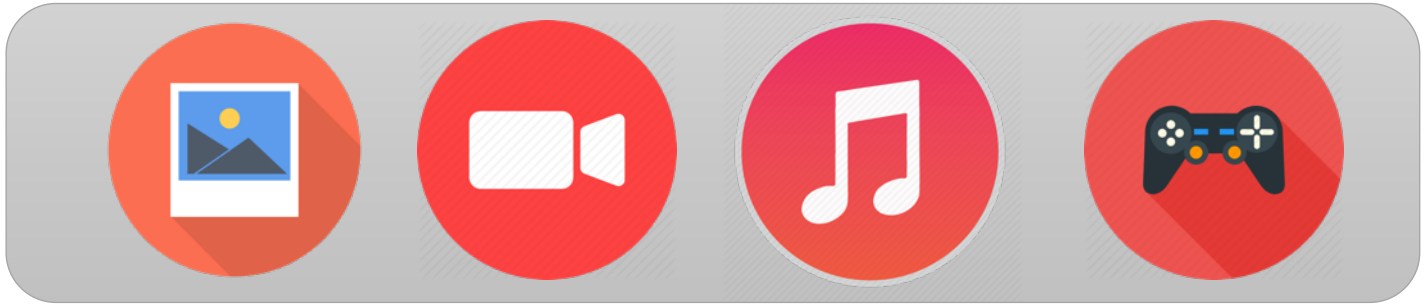
Christian Millán

1. Introduction

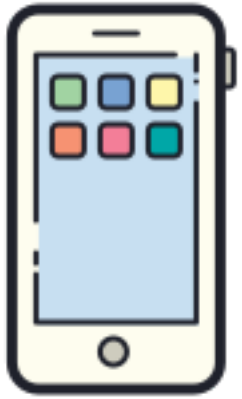
1. Introduction

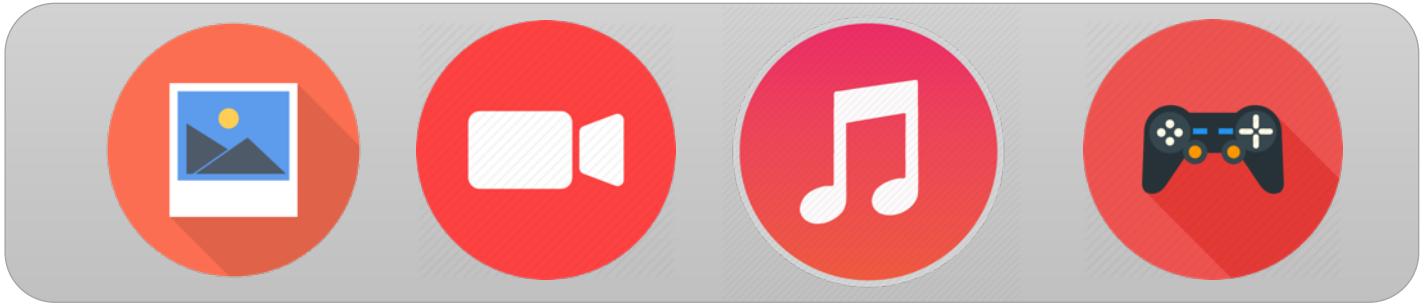
1. Introduction





1. Introduction





1011100101

1. Introduction

Programming language

1. Introduction

8086 INSTRUCTION SET

OPCODE		DESCRIPTION
AAA		ASCII adjust addition
AAD		ASCII adjust division
AAM		ASCII adjust multiply
AAS		ASCII adjust subtraction
ADC	dt,sc	Add with carry
ADD	dt,sc	Add
AND	dt,sc	Logical AND
CALL	proc	Call a procedure
CBW		Convert byte to word
CLC		Clear carry flag
CLD		Clear direction flag
CLI		Clear interrupt flag
CMC		Complement carry flag
CMP	dt,sc	Compare
CMPS	[dt,sc]	Compare string
CMPSB	"	" bytes
CMPSW	"	" words
CWD		Convert word to double word
DAA		Decimal adjust addition
DAS		Decimal adjust subtraction
DEC	dt	Decrement
DIV	sc	Unsigned divide
ESC	code,sc	Escape
HLT		Halt
IDIV	sc	Integer divide
IMUL	sc	Integer multiply
IN	ac,port	Input from port
INC	dt	Increment
INT	type	Interrupt
INTO		Interrupt if overflow
IRET		Return from interrupt
JA	slabel	Jump if above
JAE	slabel	Jump if above or equal
JB	slabel	Jump if below
JBE	slabel	Jump if below or equal
JC	slabel	Jump if carry
JCXZ	slabel	Jump if CX is zero
JE	slabel	Jump if equal
JG	slabel	Jump if greater
JGE	slabel	Jump if greater or equal
JL	slabel	Jump if less
JLE	slabel	Jump if less or equal
JMP	label	Jump
JNA	slabel	Jump if not above
JNAE	slabel	Jump if not above or equal
JNB	slabel	Jump if not below
JNBE	slabel	Jump if below or equal
JNC	slabel	Jump if no carry
JNE	slabel	Jump if not equal
JNG	slabel	Jump if not greater
JNGE	slabel	Jump if not greater or equal
JNL	slabel	Jump if not less
JNLE	slabel	Jump if not less or equal
JNZ	slabel	Jump if not zero
JNO	slabel	Jump if not overflow
JNP	slabel	Jump if not parity
JNS	slabel	Jump if not sign
JO	slabel	Jump if overflow
JPO	slabel	Jump if parity odd
JP	slabel	Jump if parity
JPE	slabel	Jump if parity even
JS	slabel	Jump if sign
JZ	slabel	Jump if zero
LAHF		Load AH from flags
LDS	dt,sc	Load pointer using DS
LEA	dt,sc	Load effective address
LES	dt,sc	Load pointer using ES
LOCK		Lock bus
LODS	[sc]	Load string
LODSB	"	" bytes
LODSW	"	" words
LOOP	slabel	Loop
LOOPE	slabel	Loop if equal
LOOPZ	slabel	Loop if zero
LOOPNE	slabel	Loop if not equal
LOOPNZ	slabel	Loop if not zero
MOV	dt,sc	Move
MOVS	[dt,sc]	Move string
MOVSB	"	" bytes
MOVSW	"	" words
MUL	sc	Unsigned multiply
NEG	dt	Negate
NOP		No operation
NOT	dt	Logical NOT
OR	dt,sc	Logical OR
OUT	port,ac	output to port
POP	dt	Pop word off stack
POPF		Pop flags off stack
PUSH	sc	Push word onto stack
PUSHF		Push flags onto stack
RCL	dt,cnt	Rotate left through carry
RCR	dt,cnt	Rotate right through carry
REP		Repeat string operation
REPE		Repeat while equal
REPZ		Repeat while zero
REPNE		Repeat while not equal
REPNZ		Repeat while not zero
RET	[pop]	Return from procedure
ROL	dt,cnt	Rotate left
ROR	dt,cnt	Rotate right
SAHF		Store AH into flags
SAL	dt,cnt	Shift arithmetic left
SHL	dt,cnt	Shift logical left
SAR	dt,cnt	Shift arithmetic right
SBB	dt,sc	Subtract with borrow
SCAS	[dt]	Scan string
SCASB	"	" byte
SCASW	"	" word
SHR	dt,cnt	Shift logical right
STC		Set carry flag
STD		Set direction flag
STI		Set interrupt flag
STOS	[dt]	Store string
STOSB	"	" byte
STOSW	"	" word
SUB	dt,sc	Subtraction
TEST	dt,sc	Test (logical AND)
WAIT		Wait for 8087
XCHG	dt,sc	Exchange
XLAT	table	Translate
XLATB	"	"
XOR	dt,sc	Logical exclusive OR

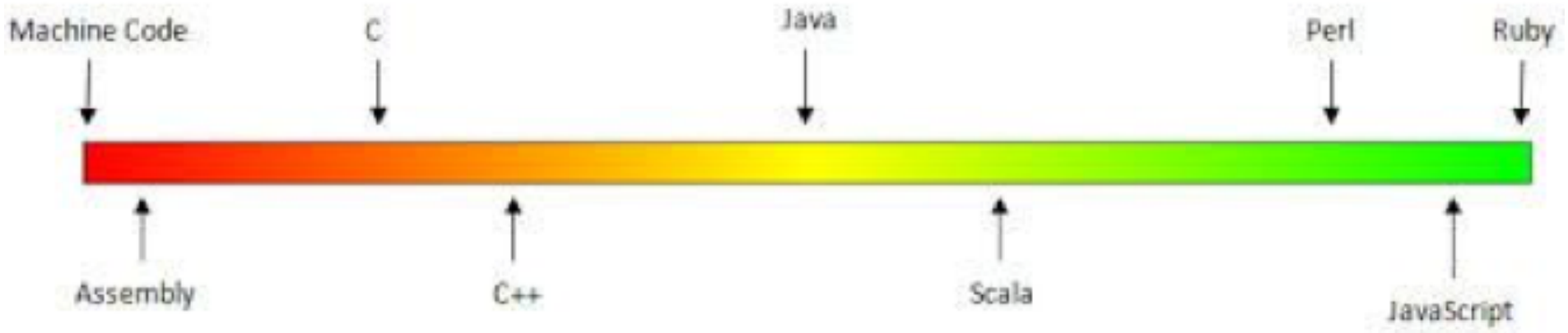
Notes:
dt - destination
sc - source
label - may be near or far address
slabel - near address

Machine language and assembler

1. Introduction

OPCODE		DESCRIPTION
AAA		ASCII adjust addition
AAD		ASCII adjust division
AAM		ASCII adjust multiply
AAS		ASCII adjust subtraction
ADC	dt,sc	Add with carry
ADD	dt,sc	Add
AND	dt,sc	Logical AND
CALL	proc	Call a procedure
CBW		Convert byte to word
CLC		Clear carry flag
CDL		Clear direction flag
CLI		Clear interrupt flag
CMC		Complement carry flag
CMP	dt,sc	Compare
CMPS	[dt,sc]	Compare string
CMPSB	"	" bytes
CMPSW	"	" words
CWD		Convert word to double word
DAA		Decimal adjust addition
DAS		Decimal adjust subtraction
DEC	dt	Decrement
DIV	sc	Unsigned divide

1. Introduction



Low-level and high-level language

WH1. Algorithms and data structures

- Edmodo

Algorithm

An algorithm is a set of instructions that solve a problem given step by step and without generating ambiguities.

1. Introduction

Method for developing an algorithm

Step 1. Obtain a description of the problem

Step 2. Analyze the problem (inputs and outputs)

Step 3. Develop a high-level algorithm

Step 4. Redefine the algorithm

Step 5. Review the algorithm (testing)

1. Introduction

Analyze the problem

- **Inputs**
  - What is needed to perform the steps?
- **Outputs**
  - ¿obtained at the end of the algorithm?
- **Data type**
  - Numbers: integer, real, complex number
  - Text: letters, words, phrases
  - Others

1. Introduction

Data types

- **System-define data types (primitives)**
  - Integer
  - Real (double, float)
  - Boolean
  - Character
- **User-defined data types**

1. Introduction

User-defined data types

- **Student**
  - Name
  - ID
  - Grade
  - Email

- **Client**
  - Name
  - ID
  - Balance
  - Address

1. Introduction

User-defined data types

- **Student**
  - Name
  - ID
  - Grade
  - Email

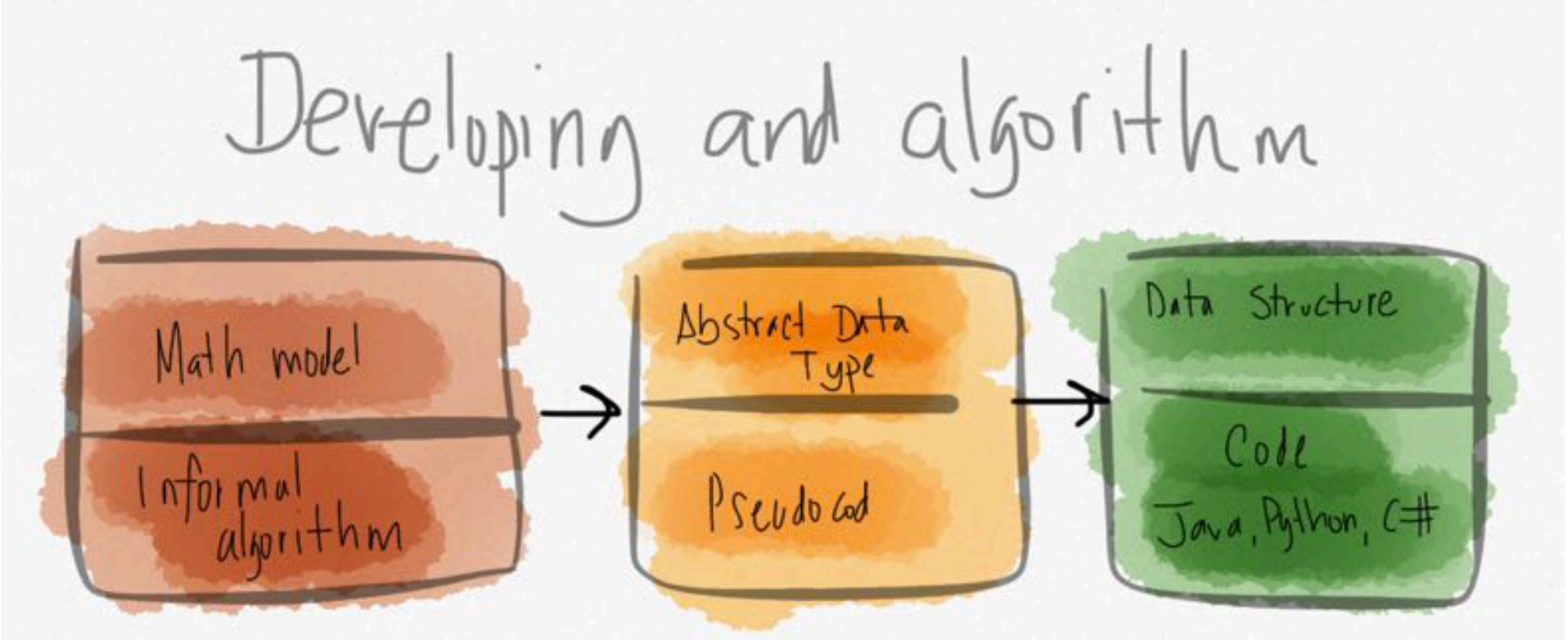
- **Client**
  - Name
  - ID
  - Balance
  - Address

C examples

```c
struct student {
    char name[50];
    char id[10];
    float grade;
    char email[50];
}
```

```c
struct client {
    char name[50];
    char id[10];
    float credit;
    char address[100];
}
```

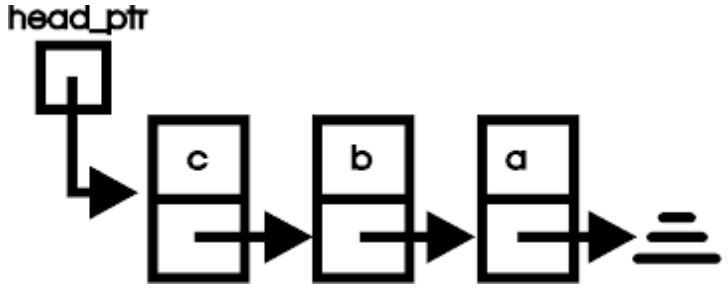
1. Introduction

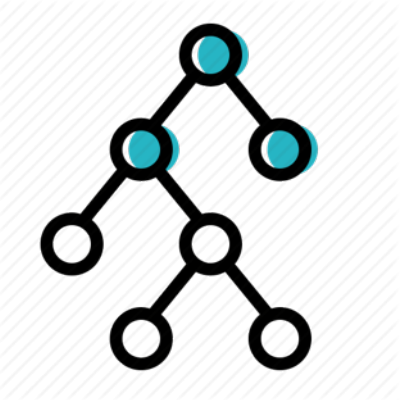


2. Data structures

Data structures classification

- Linear
- No linear





3. Abstract Data Types (ADT)

- ADT is a set of objects together with a set of operations
- ADT are mathematical abstractions nowhere in an ADT's definitions is there any mention of how the set operations is implemented.

3. Examples of ADT

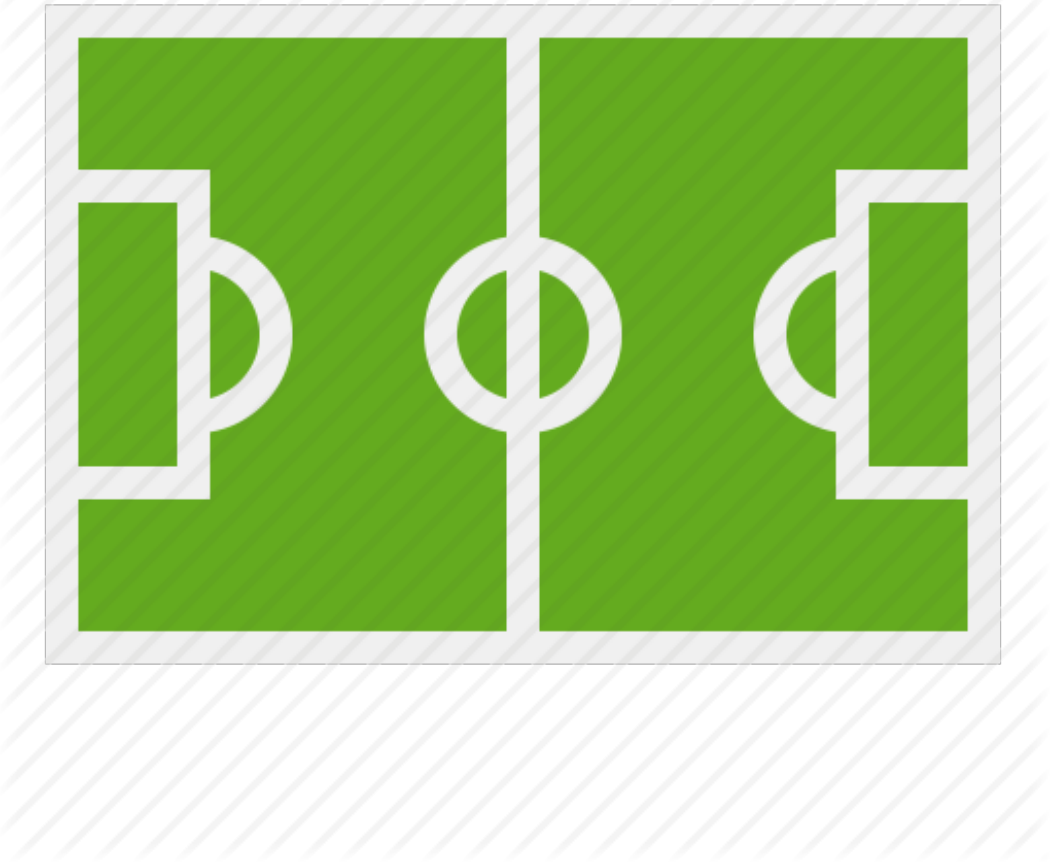
- **Linear**
  - *Linked lists*
  - *Stacks*
  - *Queues*
- **No linear**
  - *Graphs*
  - *Trees*

4. ADT graphic explanation

- List
- Stack
- Queue

5. Oriented Object Programming (OOP)

- Class
- Objects
- Heritage
- Encapsulation
- Polymorphism



WH2. ADT and OOP

- Edmodo

ACT1. Management memory

- Edmodo

6. Memory management

- Static memory
- Dynamic memory