Data Structures (Not UML)

How to feed your dragon

Amandine Decker & Marie Cousin

M1 TAL/SC 2023–2024

Université de Lorraine, LORIA

Class organisation

Two teachers :

- Amandine Decker, amandine.decker@loria.fr ;
- Marie Cousin, marie.cousin@loria.fr ;

CMs and TDs :

- 10h CM $\rightarrow$ 5 CMs of 2h each ;
- 10h TD $\rightarrow$ 5 TDs of 2h each ;
- two TD groups ;

Evaluation :

- Exam (2h) ;
- up to 2 bonus points with optional exercices (0.5 per TD) ;

What's my name ?

https ://www.creativefabrica.com/fr/product/cute-baby-black-dragon-png-file-wall-art-30/

1. Go to wooclap.com
2. Enter the event code in the top banner

Event code
ZZHMKJ

1. Send @ZZHMKJ to 06 44 60 96 62
2. You can participate

Introduction

A short history

- Comes from the latinized name of Muhammad Ibn Musa al-Khwarizmi (a 9th-century scholar, astronomer, geographer, and mathematician) who wrote about algebra ;
- But the concept is actually used since much longer (first written traces in Ancient Greece) ;

A short history

- Comes from the latinized name of Muhammad Ibn Musa al-Khwarizmi (a 9th-century scholar, astronomer, geographer, and mathematician) who wrote about algebra ;
- But the concept is actually used since much longer (first written traces in Ancient Greece) ;
- Algorithms were used to factorize, determine square roots, find prime numbers, etc. ;

A short history

- Comes from the latinized name of Muhammad Ibn Musa al-Khwarizmi (a 9th-century scholar, astronomer, geographer, and mathematician) who wrote about algebra ;
- But the concept is actually used since much longer (first written traces in Ancient Greece) ;
- Algorithms were used to factorize, determine square roots, find prime numbers, etc. ;
- But not all algorithms are mathematical ! You use algorithms every day ;
- Algorithms are basic sequences of operations that enable you to do something systematically, i.e., reach a given result once the instructions are correctly executed (cooking recipe, itinerary, assembling a piece of furniture,...).

Algorithms

Algorithm

An algorithm is a sequence of instructions which, correctly executed, leads to a given result. It may take some required information as input, and may output some data or other meaningful information after it has executed its instructions.

Algorithms

Algorithm

An algorithm is a sequence of instructions which, correctly executed, leads to a given result. It may take some required information as input, and may output some data or other meaningful information after it has executed its instructions.

Example

- A cooking recipe :
  - You need to have a sufficient amount of ingredients ;
  - it gives you specific steps to execute ;
  - for you to bake the cake you wanted to.

Algorithms

Algorithm

An algorithm is a sequence of instructions which, correctly executed, leads to a given result. It may take some required information as input, and may output some data or other meaningful information after it has executed its instructions.

Example

- A cooking recipe :
  - You need to have a sufficient amount of ingredients ;
  - it gives you specific steps to execute ;
  - for you to bake the cake you wanted to.
- The directions given by a GPS :
  - It knows your position ;
  - gives you directions to follow ;
  - for you to arrive where you wanted to.

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

$\rightarrow$ *How to feed our dragon ?*

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

$\rightarrow$ *How to feed our dragon ?*

Usually, we use algorithms to address a problem :

- what is the goal of the algorithm ?

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

$\rightarrow$ *How to feed our dragon ?*

Usually, we use algorithms to address a problem :

- what is the goal of the algorithm ?
  $\rightarrow$ *to bake a cake*

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

$\rightarrow$ *How to feed our dragon ?*

Usually, we use algorithms to address a problem :

- what is the goal of the algorithm ?
- what relevant information are in the problem ?

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

$\rightarrow$ *How to feed our dragon ?*

Usually, we use algorithms to address a problem :

- what is the goal of the algorithm ?
- what relevant information are in the problem ?
  $\rightarrow$ *the required ingredients and their quantities*

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

$\rightarrow$ *How to feed our dragon ?*

Usually, we use algorithms to address a problem :

- what is the goal of the algorithm ?
- what relevant information are in the problem ?
- how do we use these information to reach our goal ?

Problem

Problem

Intuitively, a problem is a situation that raises a question, or needs to be solved.

$\rightarrow$ *How to feed our dragon ?*

Usually, we use algorithms to address a problem :

- what is the goal of the algorithm ?
- what relevant information are in the problem ?
- how do we use these information to reach our goal ?
  $\rightarrow$ *that is what algorithms (and structures) are for :)*

Structures

Data Structure

A structure is a pre-made tool, and it organises some data

Structures

Data Structure

A structure is a pre-made tool, and it organises some data

Do not panic, it is like a toothbrush : very easy to use, and very useful !

Structures

Data Structure

A structure is a pre-made tool, and it organises some data

An algorithm deals with some data, we need to organise it.

Data Structure

A structure is a pre-made tool, and it organises some data

An algorithm deals with some data, we need to organise it.

Examples

- Ingredients are organised in containers ;
- Your flour container is most likely not of the same kind as your salt container ;
- You do not organise your plates the same way as you organise your fruits .

Structures

Data Structure

A structure is a pre-made tool, and it organises some data

An algorithm deals with some data, we need to organise it.

Examples

- Ingredients are organised in containers ;
- Your flour container is most likely not of the same kind as your salt container ;
- You do not organise your plates the same way as you organise your fruits → *Do you stack your bananas ?*.

Structures

Data Structure

A structure is a pre-made tool, and it organises some data

An algorithm deals with some data, we need to organise it.

Examples

- Ingredients are organised in containers ;
- Your flour container is most likely not of the same kind as your salt container ;
- You do not organise your plates the same way as you organise your fruits .

It is the same idea for computers and their algorithms, we need ways to organise data : data structures.

How to think algorithms

Let's sort some pancakes

What now ?

Let’s sort some pancakes

- What we have : 5 pancakes of different sizes, randomly stacked ;
- Goal : to sort them, the largest must be at the bottom of the stack, the smallest on top of it ;
- Condition : the only authorised operation is to put the shovel under a pancake and turn the stack on the shovel over.

How to think algorithms

- You just built an algorithm ! But you actually do it very often :
  - When you give the route to a tourist ;
  - When you explain to your mum where to find this paper you forgot but absolutely need ;

How to think algorithms

- You just built an algorithm ! But you actually do it very often :
    - When you give the route to a tourist ;
    - When you explain to your mum where to find this paper you forgot but absolutely need ;
- The crucial point is to give instructions that will be easily understood, *i.e.*, unambiguous instructions.

How to think algorithms

- You just built an algorithm ! But you actually do it very often :
  - When you give the route to a tourist ;
  - When you explain to your mum where to find this paper you forgot but absolutely need ;
- The crucial point is to give instructions that will be easily understood, *i.e.*, unambiguous instructions.
- Programming languages are unambiguous, as opposed to human ones ! But you have to understand how to transform your idea into a sequence of instructions the machine will understand ;

How to think algorithms

- You just built an algorithm ! But you actually do it very often :
  - When you give the route to a tourist ;
  - When you explain to your mum where to find this paper you forgot but absolutely need ;
- The crucial point is to give instructions that will be easily understood, *i.e.,* unambiguous instructions.
- Programming languages are unambiguous, as opposed to human ones ! But you have to understand how to transform your idea into a sequence of instructions the machine will understand ;
- The key here is to break your process into tiny steps, and to use data structures that are suitable to your process ! .

How to think algorithms

- You just built an algorithm ! But you actually do it very often :
  - When you give the route to a tourist ;
  - When you explain to your mum where to find this paper you forgot but absolutely need ;
- The crucial point is to give instructions that will be easily understood, *i.e.*, unambiguous instructions.
- Programming languages are unambiguous, as opposed to human ones ! But you have to understand how to transform your idea into a sequence of instructions the machine will understand ;
- The key here is to break your process into tiny steps, and to use data structures that are suitable to your process ! → *It is the point of this course !*.

Queues, Stacks

Queues

Stacks

Queues : Idea

http ://miam-images.centerblog.net

"I want to eat some cupcakes !"

- He is hungry ! We need to feed him ;
- One cupcake at a time, one after the other ;
- We have a queue of cupcakes !

Queues : FIFO

Queue

A queue is a structure containing some objects, organised one after the other. It uses the FIFO principle : First In, First Out (the first object to enter the queue will be the first to leave the queue).

Queues : FIFO

Queue

A queue is a structure containing some objects, organised one after the other. It uses the FIFO principle : First In, First Out (the first object to enter the queue will be the first to leave the queue).

$\rightarrow$ *Just like the queues you are used to in human life !*

Operations

- add an element at the end of the queue ;
- remove the first element of the queue ;
- get the first element of the queue ;
- check if the queue is empty ;
- a queue has a length ;

Remark : *Regarding the programming language you use, you do not always have access to the whole queue, but you will always have access to its head (read the object at the head of the queue).*

Queues : some formalism

If we consider a set of elements E, we can write $Queue(E)$ the set of all the queues containing elements of E. The empty queue Q_0 is in $Queue(E)$.

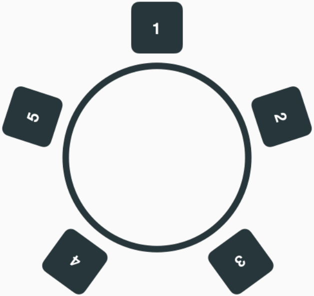
If $e \in E$, and $Q \in Queue(E)$, Q satisfies the following properties :

- $isEmpty(Q_0) = \text{True}$;
- $get(e :: Q) = e$;
- $add(e, Q) = Q :: e$
- $remove(t :: Q) = Q$
- $isEmpty(add(x, Q)) = \text{False}$.

Queues : Example

Josèphe's problem

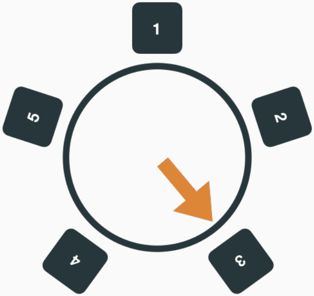
n players are sitting at a table. One of them is arbitrarily chosen as starting player. We define a step X. From this starting player (counting as 1), we count X players and eliminate the X-th one. We start again, beginning from the next one, etc. The last player in the game wins.



Queues : Example

Josèphe's problem

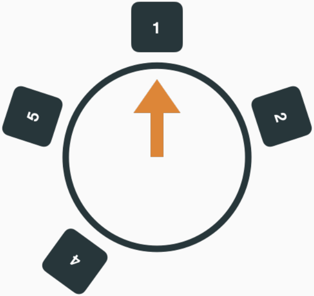
n players are sitting at a table. One of them is arbitrarily chosen as starting player. We define a step X. From this starting player (counting as 1), we count X players and eliminate the X-th one. We start again, beginning from the next one, etc. The last player in the game wins.



Queues : Example

Josèphe's problem

n players are sitting at a table. One of them is arbitrarily chosen as starting player. We define a step X. From this starting player (counting as 1), we count X players and eliminate the X-th one. We start again, beginning from the next one, etc. The last player in the game wins.



Queues : Example

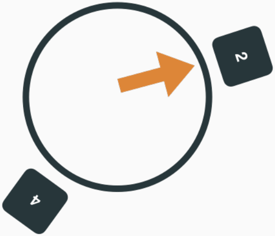
Josèphe's problem

n players are sitting at a table. One of them is arbitrarily chosen as starting player. We define a step X. From this starting player (counting as 1), we count X players and eliminate the X-th one. We start again, beginning from the next one, etc. The last player in the game wins.

Queues : Example

Josèphe's problem

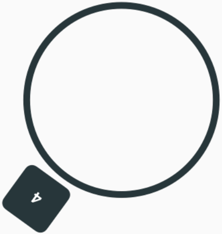
n players are sitting at a table. One of them is arbitrarily chosen as starting player. We define a step X. From this starting player (counting as 1), we count X players and eliminate the X-th one. We start again, beginning from the next one, etc. The last player in the game wins.



Queues : Example

Josèphe's problem

n players are sitting at a table. One of them is arbitrarily chosen as starting player. We define a step X. From this starting player (counting as 1), we count X players and eliminate the X-th one. We start again, beginning from the next one, etc. The last player in the game wins.

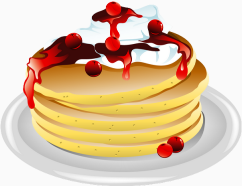


Queues : Example

Require: n, X
  $Q \leftarrow Q_0$
  for $i \in [1, n]$ **do**
    add(i, Q)
  end for
  $s \leftarrow random(1, n)$
  $current \leftarrow get(Q)$
  while $current\ ! =\ s$ **do**
    $remove(Q)$
    $add(current, Q)$
    $current \leftarrow get(Q)$
  end while
  while $!isEmpty(Q)$ **do**
    for $i \in [1, X-1]$ **do**
      $current \leftarrow get(Q)$
      $remove(Q)$
      $add(current, Q)$
    end for
    $remove(Q)$
    if $!isEmpty(Q)$ **then**
      $current \leftarrow get(Q)$
    end if
  end while
  return current

Stacks : Idea

"I want to eat some pancakes !"



http ://miam-images.centerblog.net

- He is hungry (again) ! We need to feed him ;
- One pancake at a time, but the bottom one is not (yet) accessible ;
- We have a stack of pancakes !

Stacks : LIFO

Stack

A stack is a structure containing some objects, organised one on top of the other. It uses the LIFO principle : Last In, First Out (the last object to enter the stack will be the first to leave the stack).

Stacks : LIFO

Stack

A stack is a structure containing some objects, organised one on top of the other. It uses the LIFO principle : Last In, First Out (the last object to enter the stack will be the first to leave the stack).

$\rightarrow$ *Again, just like the stacks you are used to in human life !*

Operations

- add an element on top of the stack ;
- remove the element on top of the stack ;
- get the last (= top) element of the stack ;
- check if the stack is empty ;
- a stack has a length, we can get its number of elements ;

Remark : *Regarding the programming language you use, you do not always have access to the whole stack, but you will always have access to its top (read the object on top of the stack).*

Stacks : some formalism

If we consider a set of elements E, we can write $Stack(E)$ the set of all the stacks containing elements of E. The empty stack S_0 is in $Stack(E)$.

If $e \in E$, and $S \in Stack(E)$, S satisfies the following properties :

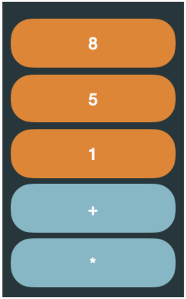
- $isEmpty(S_0) = \text{True}$
- $add(e, S) = S :: e$
- $remove(S :: e) = S$
- $remove(add(e, S)) = S$
- $isEmpty(add(x, S)) = \text{False}$
- $get(add(e, S)) = e$.

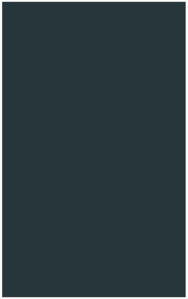
Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1 + 2$ becomes $+\ 1\ 2$;
- $4 * 5$ becomes $*\ 4\ 5$;
- $(1 + 5) * 8$ becomes $*\ +\ 1\ 5\ 8$;
- $(2 * 3) + 9$ becomes $+\ *\ 2\ 3\ 9$.



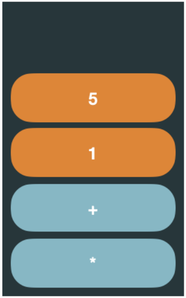


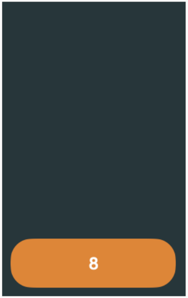
Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1 + 2$ becomes $+\ 1\ 2$;
- $4 * 5$ becomes $*\ 4\ 5$;
- $(1 + 5) * 8$ becomes $*\ +\ 1\ 5\ 8$;
- $(2 * 3) + 9$ becomes $+\ *\ 2\ 3\ 9$.



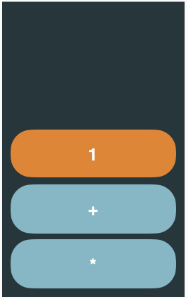


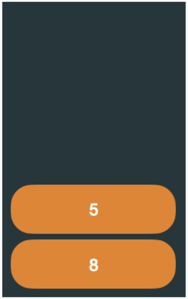
Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1 + 2$ becomes $+ \ 1 \ 2$;
- $4 * 5$ becomes $* \ 4 \ 5$;
- $(1 + 5) * 8$ becomes $* \ + \ 1 \ 5 \ 8$;
- $(2 * 3) + 9$ becomes $+ \ * \ 2 \ 3 \ 9$.



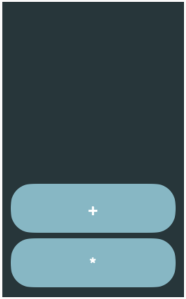


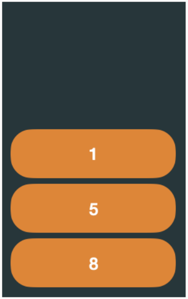
Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1 + 2$ becomes $+\ 1\ 2$;
- $4 * 5$ becomes $*\ 4\ 5$;
- $(1 + 5) * 8$ becomes $*\ +\ 1\ 5\ 8$;
- $(2 * 3) + 9$ becomes $+\ *\ 2\ 3\ 9$.



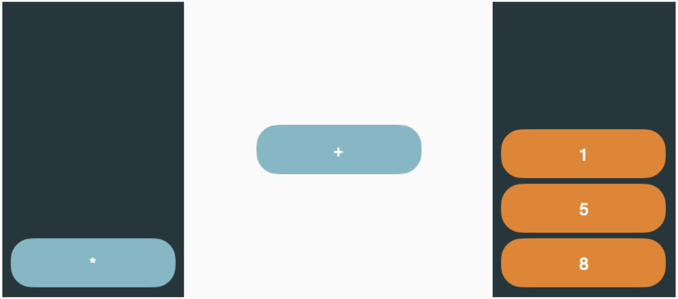


Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1 + 2$ becomes $+ \ 1 \ 2$;
- $4 * 5$ becomes $* \ 4 \ 5$;
- $(1 + 5) * 8$ becomes $* \ + \ 1 \ 5 \ 8$;
- $(2 * 3) + 9$ becomes $+ \ * \ 2 \ 3 \ 9$.



Stacks : Example

Polish writing

With this writing, you “push” the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

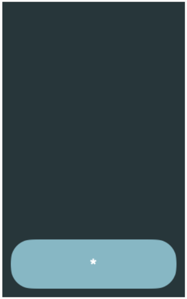
- $1 + 2$ becomes $+ \ 1 \ 2$;
- $4 * 5$ becomes $* \ 4 \ 5$;
- $(1 + 5) * 8$ becomes $* \ + \ 1 \ 5 \ 8$;
- $(2 * 3) + 9$ becomes $+ \ * \ 2 \ 3 \ 9$.

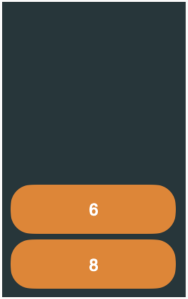
Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1 + 2$ becomes $+\ 1\ 2$;
- $4 * 5$ becomes $*\ 4\ 5$;
- $(1 + 5) * 8$ becomes $*\ +\ 1\ 5\ 8$;
- $(2 * 3) + 9$ becomes $+\ *\ 2\ 3\ 9$.





Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

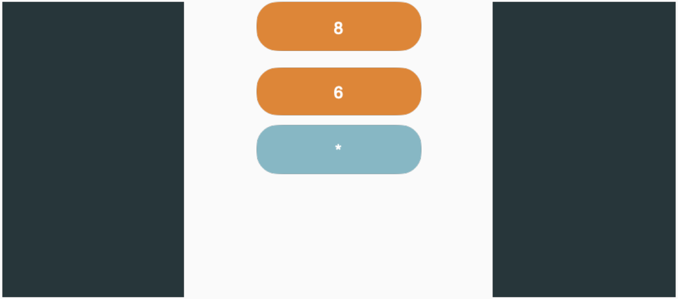
- $1+2$ becomes $+ \; 1 \; 2$;
- $4*5$ becomes $* \; 4 \; 5$;
- $(1+5)*8$ becomes $* \; + \; 1 \; 5 \; 8$;
- $(2*3)+9$ becomes $+ \; * \; 2 \; 3 \; 9$.

Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1+2$ becomes $+\ 1\ 2$;
- $4*5$ becomes $*\ 4\ 5$;
- $(1+5)*8$ becomes $*\ +\ 1\ 5\ 8$;
- $(2*3)+9$ becomes $+\ *\ 2\ 3\ 9$.

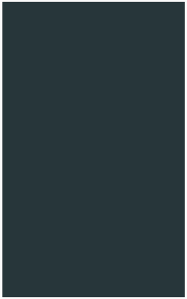


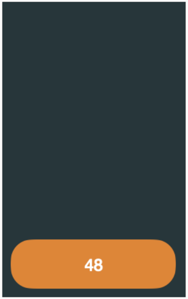
Stacks : Example

Polish writing

With this writing, you "push" the binary operator forward. With this writing, you do not need parenthesis anymore. For example :

- $1 + 2$ becomes $+ \ 1 \ 2$;
- $4 * 5$ becomes $* \ 4 \ 5$;
- $(1 + 5) * 8$ becomes $* \ + \ 1 \ 5 \ 8$;
- $(2 * 3) + 9$ becomes $+ \ * \ 2 \ 3 \ 9$.





Stacks : Example

Require: *exp*
  $S \leftarrow S_0$
  for $char \in exp$ **do**
    $add(char, S)$
  end for
  $S' \leftarrow S_0$
  while $!isEmpty(S)$ **do**
    $current \leftarrow get(S)$
    IF LOOP (right)
  end while
  return $get(S')$

IF LOOP :

if *current is an operator* **then**
  $op \leftarrow current$
  $remove(S)$
  $current \leftarrow get(S')$
  $remove(S')$
  $current \leftarrow current\ op\ get(S')$
  $remove(S')$
  $add(current, S')$
else
  $remove(S)$
  $add(current, S')$
end if

Summary

Summary

- Algorithms are sequences of instructions meant to reach a certain result ;
- You actually use them in your everyday life without necessarily realising it ;
- We use data structures to organise the data they deal with ;
- Queues and Stacks are two types of data structures that you can use for different purposes ;
- The main difference is FIFO / LIFO.