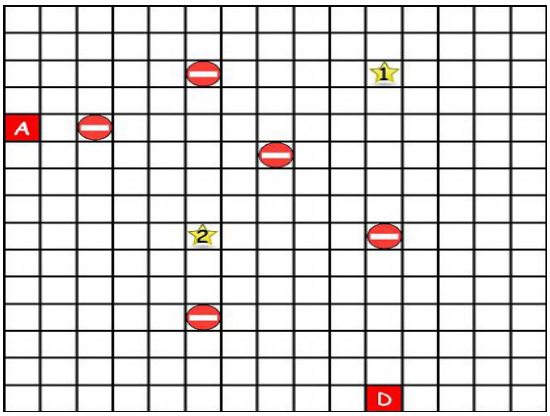
Structures de contrôle – Boucles : Algorithmes, organigrammes et programmation ARM

Boucles : algorithmes et organigrammes

Consigne :

1. ***Identifier* les notions/définitions importantes et être capable de les *expliquer*.**
2. ***Refaire les exemples* illustrant ces notions.**
3. ***les appliquer* en faisant les exercices notés exercice à faire.**

Contexte : Initiation (rappel pour certains étudiants) à l'algorithmique et notamment les structures algorithmiques de base (itérer un traitement, une action, un calcul). Ces structures sont utilisées pour résoudre des problèmes simples (trouver la valeur maximale dans un ensemble de nombres. La programmation de telles structures et leurs exécutions par un microprocesseur permettra de comprendre comment le programme est exécuté et le résultat est obtenu (tester le programme). Ces structures serviront à écrire des algorithmes (programmes associés) pour « piloter » (faire avancer) le ROBOT EVALBOT (avec différents scénarios).



Réaliser un « ROBOT nomade ». Comment le déplacer d'un point de départ vers un point d'arrivée (suivre une trajectoire), éviter des obstacles, trouver le chemin le plus court.. (euuscol.education.fr)

En partant de captures d'écran sur Google Earth en 2D, Clément Valla (architecte et designer) les modifie par des algorithmes pour obtenir des images 3D. (http://clementvalla.com/)

Acquis de formation :

- Comprendre un algorithme et/ou un organigramme et expliquer ce qu'il réalise comme traitement.
- Concevoir une solution algorithmique en réponse à un problème.
- Montrer/expliquer/argumenter que la solution proposée est correct.

1. Définitions : Algorithme :

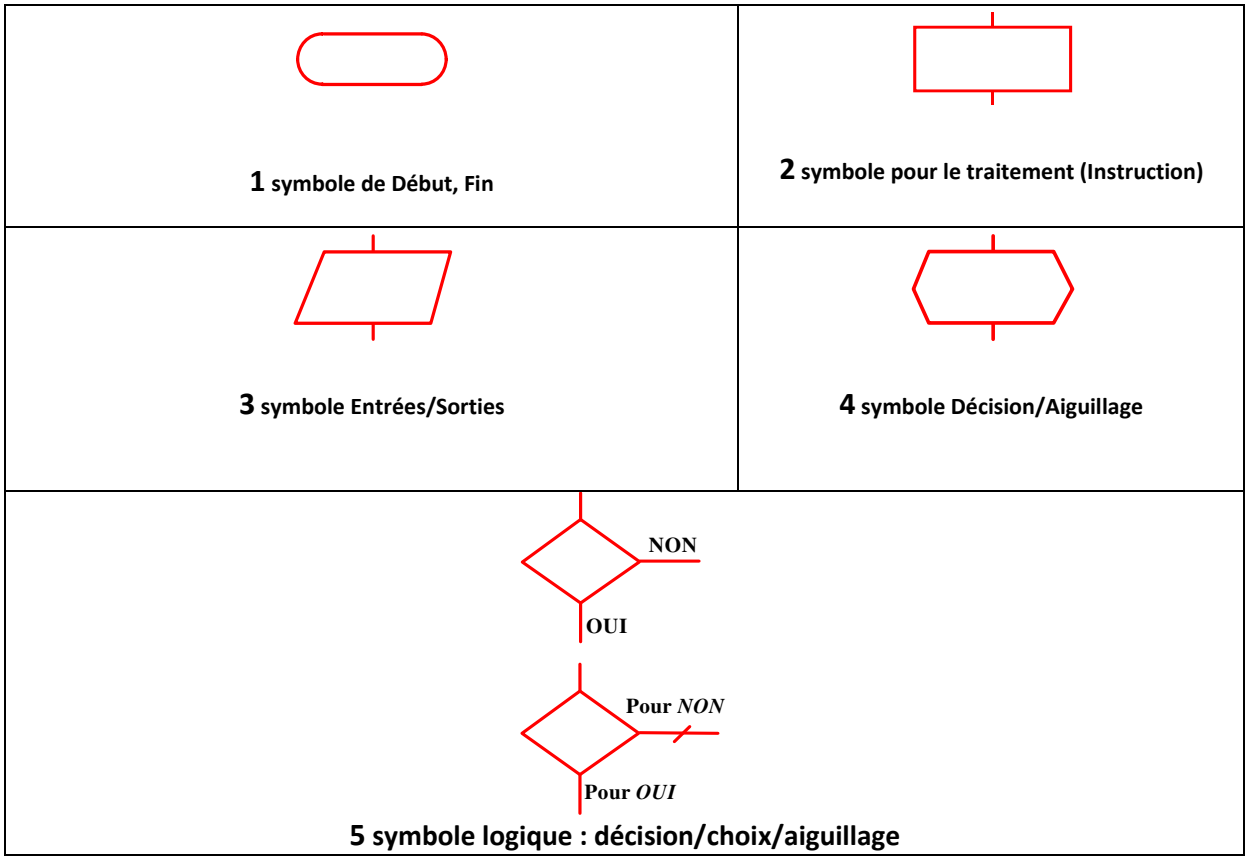
« *méthode de calcul qui indique la démarche à suivre pour résoudre une série de problèmes équivalents en appliquant dans un ordre précis une suite finie de règles* ».

Il est décrit sous la forme d'une **suite ordonnée d'instructions** laquelle définit la démarche proposée pour résoudre le problème étudié.

La complexité d'un algorithme est le nombre d'instructions élémentaires à exécuter pour résoudre le problème.

Organigramme : est une représentation graphique d'un algorithme. Il met en œuvre des **symboles** représentant des traitements, des données, des liaisons.

Il comprend un **début** et une **fin**, et permet de suivre l'ordre d'exécution des instructions de l'algorithme pour résoudre le problème étudié.



2. Les structures algorithmiques fondamentales

Les opérations élémentaires dont l'application permet de résoudre un problème peuvent être organisées en utilisant **les structures algorithmiques fondamentales** :

2.1 Structure linéaire

C'est une suite d'instructions exécutée dans l'ordre énoncé.

Exemple 1 :

$\{somme\ de\ 3\ entiers\ a, b\ et\ c\}$: ***C'est le titre de l'algorithme***

$entrées: variables\ a, b, c : entiers$

Sortie : variable somme : entier : ***somme est le nom du résultat***

$début$

$lire\ a, b, c$

$somme \leftarrow a + b + c$

fin

Affectation : le résultat de a+b+c est affecté (recopié) dans la variable appelée somme.

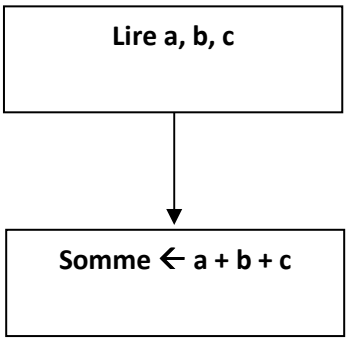
Les instructions de l'algorithme « somme de 3 entiers a, b et c» **s'exécutent les unes après les autres dans l'ordre où elles on été écrites.**

Cet algorithme est constitué d'un flux de 2 instructions

Dans cet algorithme apparaît des variables, elles permettent de donner des noms (appelés identificateurs) à des données (les variables en entrée a et b, de type entier).

La variable somme, de type entier sera affectée par le résultat de l'addition de a, b, c.

Représentation sous la forme organigramme de l'algorithme somme :



L'exécution séquentielle ci-dessous peut être rompue par des instructions conditionnelles : appelées **instructions de contrôle de flux**.

2.2 Structure alternative ou conditionnelle

Si une **condition** est **vraie**, on exécute une ou plusieurs instructions. Cette structure permet aussi de traduire des choix possibles.

si *(condition)* ***alors*** *< instruction >* ***fin si*** ***Forme réduite***	***si*** *(condition)* ***alors*** *< instruction 1 >* ***sinon*** *< instruction 2 >* ***fin si*** ***Forme complète***

Le programme teste une condition, si la condition est satisfaite (vraie), le programme exécute « instruction 1 », sinon il exécute « instruction 2 ».

Dans cette (**condition**) est une expression de type booléen comme par exemple $a < b$.

Elle est construite en général avec des comparateurs $<, =, >, \ldots$ et les opérateurs logiques ET, **OU, NON.**

Exemple 2 : Dans cette exemple, cette structure est appelée structure alternative réduite

```
{tri de 2 variables entières dans l'ordre croissant}

entrées: variables a, b : entier

sorties: variables a, b : entier

début
    si a > b alors
        echanger(a, b)
    finsi
fin
```

Dans l'algorithme de l'exemple 2, on distingue :

- **le titre** : identifie ce que fait l'algorithme
- **les déclarations :** les entrées, ce sont les objets manipulés dans le corps de l'algorithme
- **le corps de l'algorithme** : ce sont les instructions à exécuter
- et les commentaires, ils facilitent la lisibilité de l'algorithme.

Dans les déclarations, les objets peuvent être par exemples :

- **des constantes** : la valeur d'une constante ne peut être modifiée au cours de l'exécution de l'algorithme,
- **des variables** : leurs valeurs peuvent être modifiées au cours de l'exécution de l'algorithme.

Les constantes et les variables sont identifiées par leur noms, appelés **identificateurs** et leurs **types** : entier, réel, booléen, chaîne de caractères….

Dans cet exemple on distingue :

- **l'instruction qui manipule les données** : $echanger(a, b)$
- et **l'instruction** $(si\ a > b\ alors)$ **qui contrôle son exécution**.

Exemple 3 : dans cet exemple, la structure alternative est dite complète

$\{Calcul\ du\ maximum\ entre\ 2\ entiers\ a, b\}$

$entrées: variables\ a, b : entiers$

Sortie : variable maximum : entier

$début$

$si\ a < b\ alors$

$maximum \leftarrow b$

$sinon$

$maximum \leftarrow a$

$finsi$

fin

2.3 Structure répétitive, structure itérative ou boucle

Les itérations permettent d'écrire des programmes qui exécutent plusieurs fois les mêmes instructions.

Elles permettent de faire des récurrences ou de traiter des volumes importants de données.

Les programmes itératifs sont liés à la **notion mathématique de récurrence**, par exemple la suite de Fibonacci est la suite de nombres définie par la récurrence suivante :

$f_0 = f_1 = 1$

et pour $n \geq 2$ *par* $f_n = f_{n-1} + f_{n-2}$

2.3.1 la boucle - Tant que …. faire

Séquence d'instructions 1

tant que (condition) /*la condition s'appelle une condition d'arrêt*/

Faire « séquence d'instruction à répéter »

fin tant que

Séquence d'instructions2 /* quand la condition devient fausse, le programme exécute la séquence d'instructions 2*/

Exemple 4 : calcul de la partie entière inférieure de la racine carrée d'un entier donné, il permet de calculer la racine entière d'un naturel.

{calcul de la racine entière}

entrée: n : naturel

Sortie : variable racine : entier

début

$racine \longleftarrow 1$

tant que $racine \times racine \leq n$ ***faire***

$racine \longleftarrow racine + 1$

fin tant que

fin

Remarque :

la vérification de la condition est faite en début de la boucle, le corps de la boucle n'est exécuté que si la condition est vraie.

Dans la boucle **faire** … **tant que** (voir ci-dessous), la condition est vérifiée en fin de boucle. ***Le corps de boucle est exécuté au moins une fois***.

$\boldsymbol{faire}$

$a \longleftarrow a + 1$

$\boldsymbol{tant\ que}\ (a \leq n)$

2.3.2 La boucle - répéter … jusqu'à

Exemple 5 : calcul de la multiplication de 2 entiers $a \times b$ par additions successives

$\{multiplication\ de\ 2\ entiers\ a, b\ par\ additions\ successives\}$

$entrées: variables\ a, b : entiers$

Sortie : variable produit : entier

$\boldsymbol{début}$

$produit \leftarrow 0$

$\boldsymbol{repeter}$

$produit \leftarrow produit + b$

$a \leftarrow a - 1$

$\boldsymbol{jusqu'à}\ a = 0$

$\boldsymbol{fin}$

Remarque :

La vérification de la condition se fait en fin de boucle.

Le corps de la boucle est exécuté au moins une fois.

2.3.3 La boucle - Pour appelée aussi boucle for

Exemple 6 : calcul de la multiplication de 2 entiers positifs a et b donnés en utilisant l'addition entière.

$\{calcul\ de\ a \times\ b\ en\ utilisant\ l'addition\ entière\}$

$entrées: variables\ a, b : entiers$

Sortie : variable produit : entier

début

$produit \leftarrow 0$

pour $i \leftarrow 1$ ***à*** a ***faire***

$produit \leftarrow produit + b$

Fin pour

fin

Remarque :

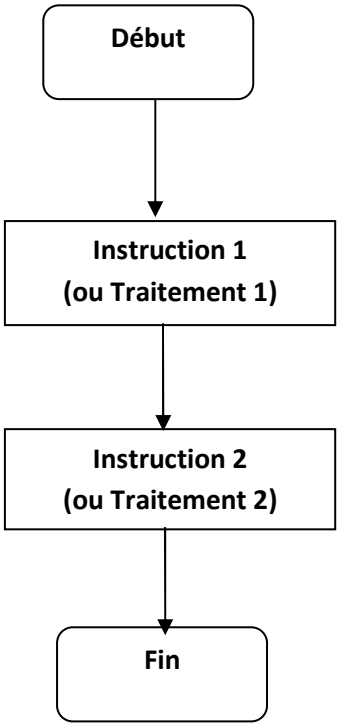
Règle à respecter : il est interdit de modifier la variable de boucle dans le corps de la boucle. Cette variable est automatiquement incrémentée à chaque cycle**.**

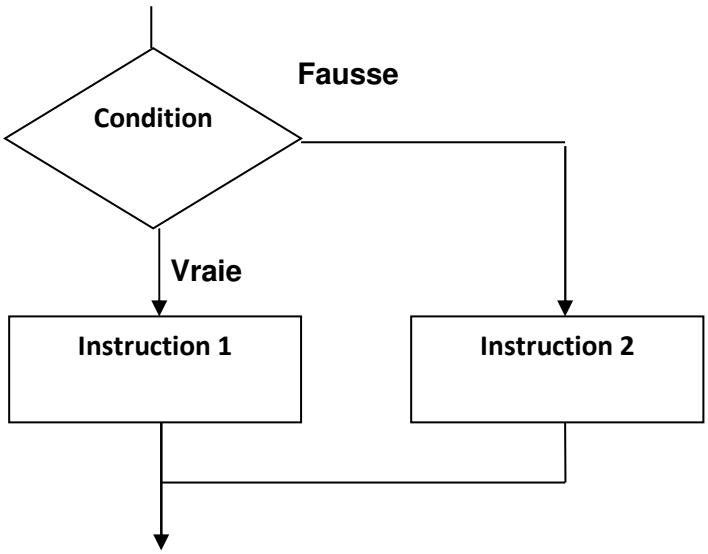
Le nombre d'exécution de la boucle est : (valeur finale de cette variable – valeur initiale) +1.

A ce stade vous êtes capable de :

- Expliquer les structures algorithmiques fondamentales : structure linéaire, structure alternative, structure itérative
- Expliquer ce que fait un algorithme
- Utiliser ces structures
- Vérifier la condition de fin de boucle
- Proposer une solution à un problème sous la forme d'un algorithme
- Illustrer le déroulement de l'algorithme proposé
- Illustrer par des exemples le résultat rendu

3. Représentation sous formes d'organigrammes des structures algorithmiques fondamentales



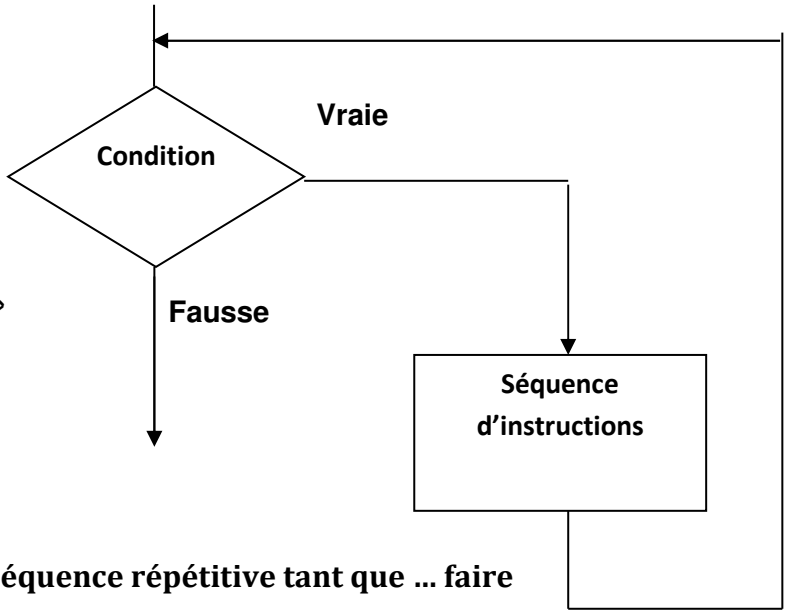


Structure alternative :

si ---- alors ----

sinon -----

fsi

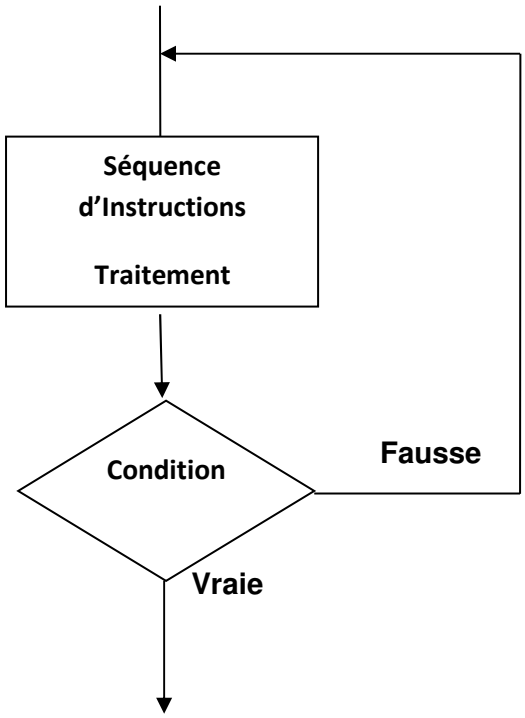


Tant que « condition »

Faire traitement

Fin de tant que

Séquence répétitive tant que … faire



Séquence répétitive répéter … jusqu'à …

répéter « traitement »

jusqu'à condition

A ce stade vous êtes capable de :

- Représenter sous la forme d'un organigramme chacune des structures algorithmiques fondamentales
- Représenter sous la forme d'un organigramme la solution à un problème

Structures de contrôle – Boucles et programmation ARM

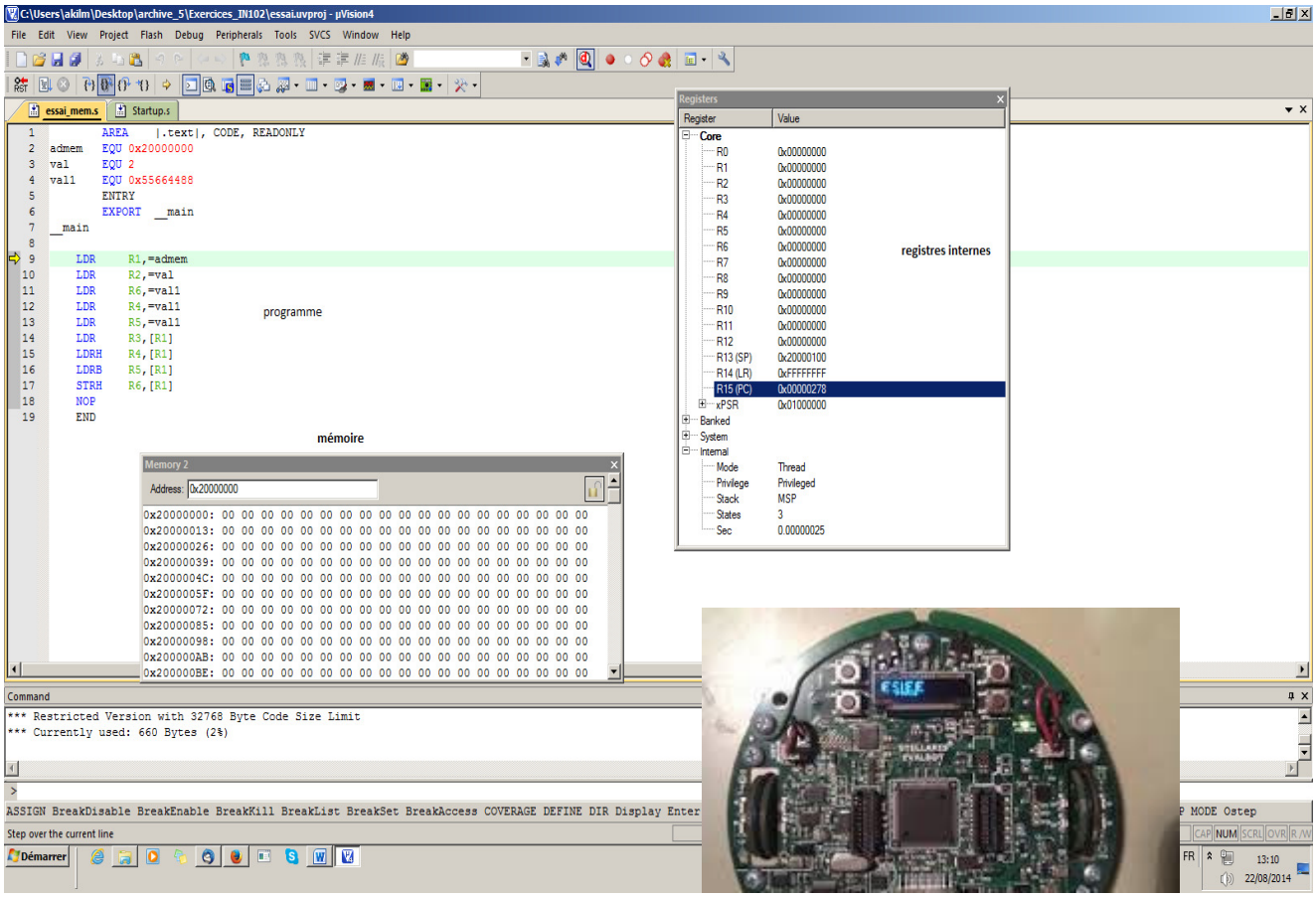
Consigne :

1. Identifier les notions/définitions importantes et être capable de les expliquer.

2. Refaire les exemples illustrant ces notions.

3. et les appliquer en faisant les exercices.

Contexte : Exécution d'un programme par un microprocesseur : exécution Instruction par Instruction, observation des registres internes, de la mémoire, des résultats obtenus (à différentes étapes d'un programme), tests et corrections. Ecriture de programmes, mise en œuvre de ces programmes en utilisant l'environnement de simulation KEIL - uVISION 4. Cette phase est importante pour programmer le ROBOT EVABOT, afin de le « piloter » (faire avancer avec différents scénarios : suivre une trajectoire, détecter des obstacles, changer d'orientation, etc..).



Acquis de formation :

- Ecrire des programmes avec le langage ARM Cortex M3
- Utiliser les instructions de branchement conditionnel pour écrire des programmes comportant :
  - Des structures alternatives
  - Des structures itératives
- Expliquer le déroulement d'un programme et le (les) résultat(s) rendu(s)
- Corriger un programme

Cette partie est consacrée à la programmation en langage d'assemblage ARM Cortex – M3 des structures de contrôle et des boucles : Ecrire des programmes.

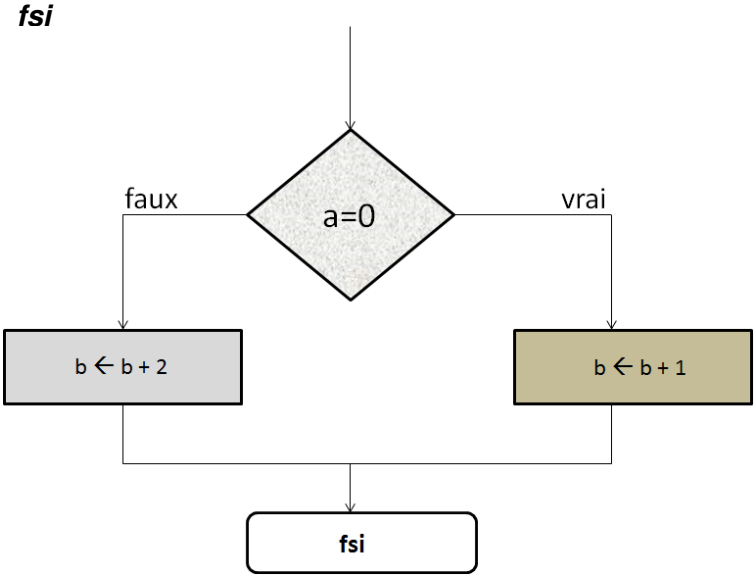
Elle répond à la question :

Comment écrire le programme correspondant à une des structures algorithmiques traitées dans la première partie ?

L'exemple suivant (même s'il est très simple), vous propose une méthode (démarche) pour répondre à cette question.

Exemple : Soit 2 variables a et b $\in N$ (entiers naturels), on suppose que ces variables sont rangées temporairement dans les registres R1 et R2 (respectivement pour a et b).

```
si      a = 0       alors
                        b ←b+1              /* Instruction 1*/
                    sinon
                        b← b+2              /*Instruction 2*/
fsi
```



- On doit d'abord tester si a = 0
- Si a = 0 l'instruction 1 est exécutée.
- Si a $\neq 0$ l'instruction 2 est exécutée

Comment un microprocesseur teste si un résultat est égal à 0 ?

Comme la variable a est (affectée) dans R1, il faut tester si R1=0, or cette condition est vérifiée si l'indicateur Z = 1.

Si Z = 1, le microprocesseur exécute l'instruction 1, sinon il exécute l'instruction 2.

Les instructions de **branchement** (*aller exécuter*) **conditionnel** (si la condition spécifiée) permettent de tester la condition indiquée et en fonction de l'état de la condition testée (vraie ou faux) *aller exécuter* l'instruction correspondante (ou la séquence d'instructions).

Syntaxe de l'instruction de **B**ranchement conditionnel

B<c> **<labelt>**

Le champ <label> (étiquette = nom) identifie l'instruction à exécuter si la condition spécifiée (indiquée) par le champ <c> est vraie.

<c> désigne donc le suffixe de la condition à tester.

Ainsi le suffixe pour tester si Z = 1 (EQual) est EQ

```
        BEQ         Inst1       /* aller à l'étiquette Inst1 si la condition EQ est vraie*/
        |
        Instruction 1 (ou séquence 1 d'Instructions)
        |
        B           fsi         /* branchement inconditionnel à l'étiquette fsi*/
Inst1   Instruction 2 (séquence 2 d'instructions)
        |
 fsi    Instruction 3
        ......
        ....
```

Remarque : si Z = 1 seule l'instruction 1 (ou la séquence 1 d'instructions) doit être exécutée, il faut donc mettre après l'instruction 1 (la séquence 1 d'instructions) l'instruction de *Branchement inconditionnel* **(B fsi**) pour aller exécuter l'instruction 3 (séquence 3 d'instructions) indiquée par l'étiquette fsi.

Pour tester si R1 = 0, on peut par exemple le comparer à 0, à l'aide de l'instruction de comparaison :

```
CMP   R1,#0
```

Cette instruction consiste à faire l'opération (R1 – 0) et à positionner les indicateurs dont l'indicateur Z, si celui-ci est égal à 1 alors R1=0.

Une instruction de branchement conditionnelle permet si la condition spécifiée est vraie d'aller exécuter l'instruction indiquée par l'étiquette (label).

```
        CMP   R1,#0
        BEQ   Inst1          /* aller à l'étiquette Inst1 si la condition EQ est vraie*/
        ADD   R2,#2
        B     fsi            /* branchement inconditionnel à l'étiquette fsi*/
Inst1   ADD   R2,#1
fsi
```

Important :

Les instructions de contrôle changent la valeur de PC (le registre Program Counter contenant l'adresse de l'instruction à exécuter, c'est-à-dire le registre R15).

Ainsi, lors de l'exécution de l'instruction BEQ inst1

- le registre PC indique l'adresse l'instruction suivante : c'est-à-dire ADD R2,#2, que nous appelons PC courant.

Si la condition est vraie, PC change de valeur et devient = PC courant + Inst1.

Inst1 définit le déplacement pour atteindre l'instruction indiquée par l'étiquette Inst1.

Ainsi le registre PC (registre R15) = adresse (indique) l'instruction spécifiée par l'étiquette Inst1.

Cette étiquette code le déplacement (saut) à effectuer de l'adresse courante pour atteindre l'instruction indiquée par l'étiquette Inst1.

Syntaxe de l'instruction de branchement conditionnelle

B<c> **<labelt>** label (étiquette) identifie l'instruction à exécuter si la condition est vraie. <c> désigne le suffixe de la condition à tester :

Suffix	Flags	Meaning
EQ	Z set	Equal
NE	Z clear	Not equal
CS or HS	C set	Higher or same (unsigned >=)
CC or LO	C clear	Lower (unsigned <)
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned >)
LS	C clear or Z set	Lower or same (unsigned <=)
GE	N and V the same	Signed >=
LT	N and V differ	Signed <
GT	Z clear, N and V the same	Signed >
LE	Z set, N and V differ	Signed <=
AL	Any	Always. This suffix is normally omitted.

Syntaxe de l'instruction de comparaison

CMP **<R_n>,#<const>** compare et positionne les indicateurs

Soit à comparer deux 2 entiers a et b :

CMP a,b

Suffixe – du code condition : <c>	a et b (entiers naturels)	a et b (entiers)
a = b	EQ	EQ
a ≺ b	LO	MI
a ≤ b	LS	LE
a ≻ b	HI	GT
a ≥ b	HS	GE

A ce stade vous êtes capable de :

- Expliquer ce que fait une instruction de branchement conditionnel
- Utiliser ces instructions pour écrire des programmes comportant des structures : alternatives et de boucles
- Utiliser une instruction de comparaison
- Expliquer la différence entre une instruction de branchement conditionnel et une instruction de branchement inconditionnel
- Expliquer l'évolution du registre PC