MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE UNIVERSITÉ MOULOUD MAMMERI TIZI-OUZOU



FACULTÉ DE TECHNOLOGIE
DÉPARTEMENT TECHNOLOGIE

Présentée par

Dr. Mohamed DJOUDER

Cours

Structure des Ordinateurs et Applications

SOMMAIRE

II. CHAPITRE II: NOTION SUR L'ALGORITHMIQUE ET LA PROGRAMMATION EN LANGAGE C. 59

II.1 Objectifs du chapitre 59

II.2 Qu'est-ce qu'un Algorithme ? 59

II.3 . Démarche d'analyse d'un problème 60

II.4 Notion d'identificateur 61

II.5 Constantes et variables 61

II.6 Types de données 62

II.7 Structure d'un algorithme / programme 62

II.8 Corps 64

II.9 . Instructions d'Entrées/Sorties (Lecture / Écriture) 64

II.10 Instruction d'affectation 65

II.11 Structures de contrôle conditionnelle 66

II.11.1 Test alternatif simple 66

II.11.2 Test alternatif double 67

Exemple sur les contrôles conditionnelles 67

II.12 Structures de contrôle répétitives 67

II.12.1 Boucle Pour (for) 68

II.12.2 Boucle Tant-que (while) 68

II.12.3 Boucle Répéter (Repeat) 68

II.

CHAPITRE II: NOTION SUR L'ALGORITHMIQUE ET LA PROGRAMMATION EN LANGAGE C

II.1 Objectifs du chapitre

- Dans ce chapitre, vous allez voir :
- Les notions de base sur les sous-programmes.
- Structure générale d'un programme C.
- Déclaration de sous-programme :
- Procédure & Fonction et notions de paramètres formels
- Appel à sous-programme et notion de paramètres effectif
- Transmission et passage de paramètre : passage par valeur et passage par variable
- Exercices d'Applications

II.2 Qu'est-ce qu'un Algorithme ?

Un algorithme est une procédure permettant de résoudre un problème en décrivant précisément les opérations à effectuer. Résoudre un problème implique généralement de transformer des données d'entrée en données de sortie.

• Les données d'entrée peuvent être soit entièrement connues avant d'exécuter l'algorithme, soit fournies au cours de son exécution.

• De même, les données de sortie sont soit générées en fin d'exécution, soit communiquées en cours d'exécution de l'algorithme.

Autre définition de l'Algorithme : La définition informelle du terme algorithme : procédure de calcul bien définie qui prend en entrée une valeur, ou un ensemble de valeurs, et qui donne en sortie une valeur, ou un ensemble de valeurs. Un algorithme est donc une séquence d'étapes de calcul qui transforment l'entrée en sortie. L'on peut aussi considérer un algorithme comme un outil permettant de résoudre un problème de calcul bien spécifié. L'énoncé du pro-

blème spécifie, en termes généraux, la relation désirée entre l'entrée et la sortie. L'algorithme décrit une procédure de calcul spécifique permettant d'obtenir cette relation entrée/sortie

L'implémentation d'un algorithme :

Un algorithme peut être implémenté sous la forme d'un programme, qui spécifie comment il peut être exécuté par un processeur d'un ordinateur.

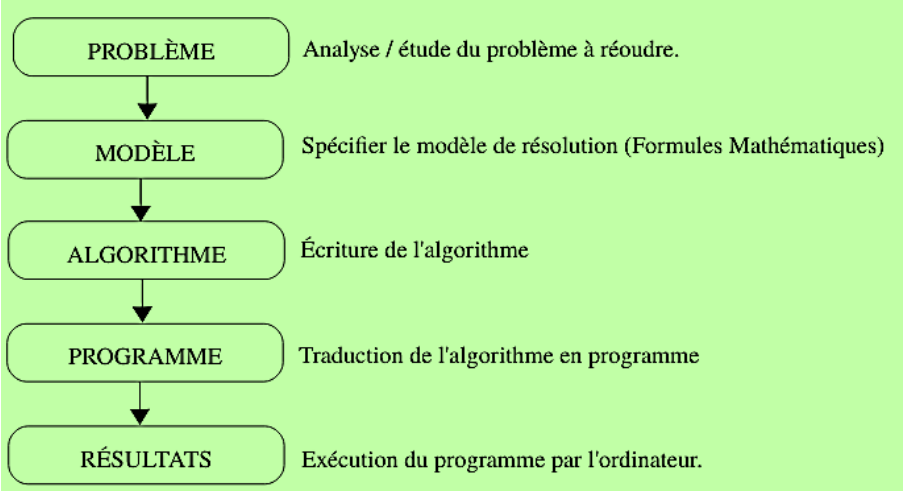
- L'ensemble des opérations élémentaires composant les programmes et pouvant être exécutées par un processeur est limité.
- Cet ensemble dépend de l'architecture matérielle du processeur utilisé.
- Il a été établi qu'un très petit ensemble d'opérations suffit à implémenter n'importe quel algorithme.

Exemples de problèmes :

- Calculer la racine carrée d'un nombre positif, a une certaine précision prés.
- Déterminer si un nombre naturel est premier ou non.
- Calculer le plus court chemin entre deux points dans un espace donné.
- Affronter un adversaire au jeu d'échecs.
- Repérer la position d'un visage dans une image.
- Traduire une conversation du français vers l'anglais.
- Simuler l'évolution d'un phénomène physique ...etc

II.3 . Démarche d'analyse d'un problème

Comme vu dans le point précédent, un algorithme représente une solution à un problème donné. Pour atteindre cette solution algorithmique un processus d'analyse et de résolution sera appliqué. Ce processus est constitué des étapes suivantes :



II.4 Notion d'identificateur

Un identificateur est une chaîne de caractères contenant uniquement des caractère alphanumériques (alphabétiques de [a-z] et [A-Z] et numérique [0-9]) et tiré 8 '_' (trait souligné), et qui doit commencer soit par un une lettre alphabétique ou _∗. Un identificateur permet d'identifier d'une manière unique un algorithme (ou un programme), une variable, une constante, une procédure ou une fonction.

Dans un langage de programmation donnée, on n'a pas le droit d'utiliser les mots réservés (mots clés) du langage comme des identificateurs.

Parmi les mots clés du langage C: « auto, break, case, char, continue, do, double, else, extern, float, for, goto, if, int, long, return short, sizeof, static, struct, switch, typedef, union, unsigned, while »

a1 : est un identificateur valide.
a_1 : est un identificateur valide.
A_1 : est un identificateur valide.
x12y : est un identificateur valide.
x1 y : est un identificateur non valide (à cause du blanc ou l'espace).
x1-y : est un identificateur non valide (à cause du signe -).
x1_y : est un identificateur valide.
1xy : est un identificateur non valide (commence un caractère numérique)

II.5 Constantes et variables

Les données manipulées par un algorithme (ou un programme) sont soit des constantes ou des variables :

Les Constantes

Une constante est un objet contenant une valeur qui ne peut jamais être modifiée. Son objectif est d'éviter d'utiliser une valeur d'une manière directe. Imaginons qu'un algorithme utilise la valeur 3.14 une dizaine de fois (le nombre d'occurrences de la valeur 3.14 est par exemple 15) et qu'on veut modifier cette valeur par une autre valeur plus précise : 3.14159. Dans ce cas on est amené à modifier toutes les occurrences de 3.14. Par contre, si on utilise une constante « PI = 3.14 » on modifier une seule fois cette constante.

Les Variables

Une variable est un objet contenant une valeur pouvant être modifiée. Toute les données (variable ou constante) d'un algorithme possède un type de données (domaine de valeurs possibles).

Exemples de Déclarations de constantes :

- ***CONST PI = 3.14 ; {constante réelle}***
- ***CONST CHARGE = 1.6E-19; {charge de l'électron}***
- ***CONST MASSE = 0.9E-27; {masse de l'électron}***
- ***CONST message = 'Bonjour tout le monde'; {constante chaîne de caractère}***

II.6 Types de données

Dans l'algorithmique, nous avons cinq types de base :

- ***Entiers : représente l'ensemble {..., -4, -3, -2, -1, 0, 1, 2, 3, 4, ...}***
- ***Réels : représente les valeurs numériques fractionnels et avec des virgule fixes (ou flottante)***
- ***Caractères : représente tous les caractères imprimables.***
- ***Chaînes de caractères : une séquence d'un ou plusieurs caractères***
- ***Booléens (logique) : représente les deux valeurs « TRUE » et « FALSE ».***

Le tableau suivant montre la correspondance entre les types d'algorithme et les types du langage de programmation C.

Algorithme	En C
Entier	int
Réel	float
Booléen	bool
Caractère	char
chaîne	Tableau de char (char[])

II.7 Structure d'un algorithme / programme

Un algorithme manipule des données, les données avant de les utiliser il faut les identifier et les déclarer en utilisant les identificateurs.

Un algorithme est constitué de trois parties :

- ***Entête :*** dans - cette partie on déclare le nom de l'algorithme à travers un identificateur.
- ***Déclarations :*** dans cette partie on déclare toutes les données utilisées par l'algorithme.
- ***Corps :*** représente la séquence d'actions (instructions)

Pour écrire un algorithme, il faut suivre la structure suivante :

Syntaxe de l'entête en algorithme et en langage C

```
Algorithme <id_algorithme>        #include <stdio.h>
     <Déclarations>               int main()
Début                             {
     <Corps : Instructions>            <déclarations>;
Fin.                                   <Instructions>;
                                       return 0;
                                  }
```

Remarque

Pour commenter un code écrit en programme C, on écrit les commentaires entre /* ... */.

Par exemple :

/* Ceci est un commentaire */ à plusieurs lignes

On peut aussi utiliser les commentaire ligne

// Ceci est un commentaire ligne

- En langage C, la partie déclaration peut être avant la fonction main, comme elle peut être dans la fonction main.

Exemple de déclaration des Variables :

En Algorithme :

- x : réel ; {est une variable réelle.}
- n, m : entier ; {sont deux variables entières.}
- s : chaîne char ; {est une variable chaîne de caractères}
- b1, b2, b3 : booléen ; {sont des variables chaîne de booléennes}
- c1 : caractère ; {est une chaîne de caractères}

En langage C :

- float x; {variable réelle.}
- intx,y; {deux variables entières.}
- s[100]; {variable chaîne de caractères}
- bool b1, b2, b3; {#include}
- char c1; {variable caractère}.

II.8 Corps

Le corps d'un algorithme est constitué d'un ensemble d'actions / instructions séquentiellement et logiquement ordonnées. Les instructions sont de cinq types, à savoir :∗

- **Lecture :** L'opération de faire entrer des données à l'algorithme. Une lecture consiste à donner une valeur arbitraire à une variable.
- **Écriture :** L'opération d'affichage de données. Elle sert à afficher des résultats ou des messages.
- **Affectation :** ça sert à modifier les valeurs des variables.
- **Structure de contrôle :** Elle permet modifier la séquentialité de l'algorithme, pour choisir un chemin d'exécution ou répéter un traitement.
- **Structure de Test alternatif simple / double**
- **Structures répétitives (itératives) :**
  - ***La boucle Pour***
  - ***La boucle Tant-que***
  - ***La boucle Répéter***

Attention :

Dans le langage C, chaque instruction se termine par un point-virgule.

II.9 . Instructions d'Entrées/Sorties (Lecture / Écriture)

Les Entrées (Lecture)

Une instruction d'entrée nous permet dans un programme de donner une valeur quelconque à une variable. Ceci se réalise à travers l'opération de lecture.

Les Sorties (Écriture)

Une instruction de sortie nous permet dans un programme d'afficher un résultat (données traitées) ou bien un message (chaîne de caractères). Ceci se réalise à travers l'opération d'écriture.

La syntaxe et la sémantique d'une lecture et d'une écriture sont comme suit :

Les Entrées		
Algorithme	**Langage C**	**Signification**
Lire()	scanf("%c" ,);	Donner une valeur quelconque à la variable dont l'identifiant .
Lire(, , …);	; scanf("%c %c %c", , , …);	Donner des valeurs aux variables , , etc.
Les Sorties		
Algorithme	**Langage C**	**Signification**
Écrire(<id_var> | <id_const> | <valeur>, <expression>);	printf("%f", <id_var> | <id_const> | <valeur>, <expression>);	Afficher une valeur d'une variable, d'une constante, valeur immédiate ou calculée à travers une expression.

Exemple d'écriture et de la lecture

En Algorithme :

- **Ecrire('Bonjour') ; {afficher le message Bonjour}**
- **Ecrire(a, b, c) ; {afficher les valeurs des variables a, b et c}**
- **Ecrire(5+2) ; {afficher le résultat de la somme de 5 et 2 : afficher 7}**
- **Ecrire('La valeur de x : ', x); {afficher le résultat de la variable x}**

En langage C :

- **printf ("Bonjour"); {afficher le message Bonjour}**
- **printf("%d %d %d "a, b, c); {afficher les valeurs des variables a, b et c}**
- **printf("%f", 5+2); {afficher le résultat de la somme de 5 et 2 : afficher 7}**
- **printf("La valeur de x : %d", x); {afficher le résultat de la variable x}**

II.10 Instruction d'affectation

Une affectation consiste à donner une valeur (immédiate, constante, variable ou calculée à travers une expression) à une variable. La syntaxe d'une affectation est :

En algorithmique :

$$Id - Variable \leftarrow Valeur\ ;$$
$$Id - variable \leftarrow Expression\ ;$$

En langage C :

$$Id - Variable = Valeur\ ;$$
$$Id - variable = Expression\ ;$$

Exemple

En Algorithme :

$$a \leftarrow 5\ ;$$
$$b \leftarrow a + 5\ ;$$
$$Supp \leftarrow a > b\ ;$$

En langage C :

```
a = 5; {mettre la valeur 5 dans la variable a}
b = a+5; {mettre la valeur de l'expression a+5 dans la variable B}
Supp = a>b; {a>b donne un résultat booléen, donc Supp est une variable booléenne}
```

II.11 Structures de contrôle conditionnelle

Ces structures sont utilisées pour décider de l'exécution d'un bloc d'instruction : est-ce-que ce bloc est exécuté ou non. Ou bien pour choisir entre l'exécution de deux blocs différents. Nous avons deux types de structures conditionnelles :

II.11.1 Test alternatif simple

Un test simple contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est-ce-que le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on ne l'exécute pas. La syntaxe d'un test alternatif simple est comme suit :

```
si <Condition> alors
    <bloc_instructions_si>;
finsi;
```

```
if (<condition>)
{
    <bloc_instructions_if>;
}
```

Exemple :

```
lire(x);
si x > 2 alors
    x ← x + 3;
finsi
écrire (x)
```

```
scanf("%d", x);
if (x > 2)
{
    x = x + 3;
}
printf("%d", x);
```

II.11.2 Test alternatif double

Un test double contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le second. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second. La syntaxe d'un test alternatif simple est :

```
si <Condition> alors
    <bloc_inst_si>;
sinon
    <bloc_inst_sinon>;
finsi
```

```
if (<condition>)
{
    <bloc_inst_if>;
}
else
{
    <bloc_inst_else>;
}
```

Exemple sur les contrôles conditionnelles

```
lire(x)
si x > 2 alors
    x ← x + 3
sinon
    x ← x - 2
finsi
écrire (x)
```

```
scanf("%d", x);
if (x > 2){
    x:= x + 3;
}
else {
    x:= x - 2;
}
printf("%d", x);
```

II.12 Structures de contrôle répétitives

Les structures répétitives nous permettent de répéter un traitement un nombre fini de fois. Par exemple, on veut afficher tous les nombres premiers entre 1 et N (N nombre entier positif donné).

Nous avons trois types de structures itératives (boucles) :

II.12.1 Boucle Pour (for)

La structure de contrôle répétitive pour (for en langage C) utilise un indice entier qui varie (avec un incrément = 1) d'une valeur initiale jusqu'à une valeur finale. À la fin de chaque itération, l'indice est incrémenté de 1 d'une manière automatique (implicite).

La syntaxe de la boucle for comme suit :

```
pour <ind>←<vi> à <vf> faire
    <instruction(s)>
finPour;
```

```
for (<ind>:=<vi>;<ind> <= <vf>;<ind>++)
 {
     <instruction(s)>;
 }
```

II.12.2 Boucle Tant-que (while)

La structure de contrôle répétitive tant-que (while en langage C) utilise une expression logique (booléenne) comme condition d'accès à la boucle∗ : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on entre à la boucle, sinon on quitte la boucle.

La syntaxe de la boucle tant-que est comme suit :

```
tant-que <condition> faire
    <instruction(s)>
finTant-que;
```

```
while (<condition>)
 {
     <instruction(s)>;
 }
```

II.12.3 Boucle Répéter (Repeat)

La structure de contrôle répétitive répéter utilise une expression logique ou booléenne comme condition de sortie de la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) on sort de la boucle, sinon on y accède (on répète l'exécution du bloc).

La syntaxe de la boucle répéter est comme suit :

```
repéter
    <instruction(s)>
jusqu'à <condition>;
```

```
do {
    <instruction(s)>;
while (<condition2>);
```

Exercices d'applications

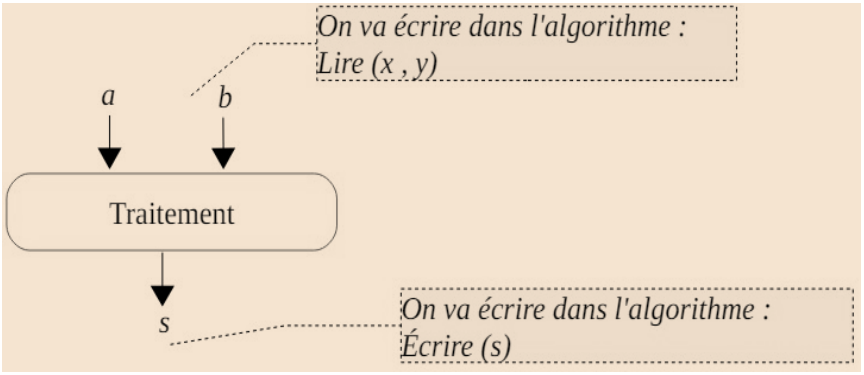
Exercice 01 : Somme de deux variables entières

1) Écrire un algorithme qui permet de réaliser la somme de deux variables entières.
2) Traduire l'algorithme en programme C

Solution :

Analyse et Discussion : On veut réaliser la somme de deux variables entières a et b, on doit tout d'abord donner deux valeurs aux deux variables (on aura besoins de deux lectures). À la fin, on doit calculer leur somme dans une troisième variable S et afficher ensuite la valeur de S.

Donc l'algorithme possède deux variables d'entrée (a et b) et une variable de sortie S. On peut schématiser l'algorithme comme suit :



Après avoir déterminé les variables d'entrée et de sortie, il nous reste à trouver l'idée du traitement. Le traitement est simple, il suffit de réaliser l'affectation suivante :

s ← a + b.

Donc l'algorithme (et sa traduction en programme C sera comme suit :

```
Algorithme somme;
      Variables
            a, b, s : entier
Début
      Lire (a, b)
      s ← a + b
      Écrire (s)
Fin
```

```c
#include <stdio.h>
int main()
{
    int a, b, s;
        //Entrée ....
    printf("Donnez la valeur de a et b : ");
    scanf ("%d %d" , &a, &b);
    //Traitement ....
    s=a+b;
    //Sortie ...
    printf("la somme = %d",s);
    return 0;
}
```

Exercice 02 : Nombre pair ou impair

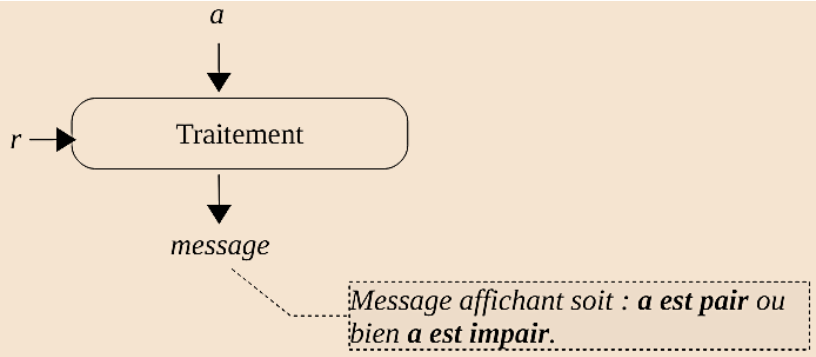
Écrire un algorithme qui permet d'indiquer si le nombre est pair ou non (un nombre pair est divisible par 2).

Traduire l'algorithme en programme C.

Solution :

Analyse et Discussion : L'algorithme prend comme donnée d'entrée une variable entière et affiche un message indiquant si ce nombre-là est pair ou non. Un nombre pair est divisible par 2.

Donc, nous avons une variable d'entrée et un message comme sortie



Pour savoir si un nombre est pair ou non, il suffit de calculer son reste de division par 2. On peut utiliser directement la fonction mod qui s'écrit : n mod b = le reste de division de n sur b.

Donc l'algorithme (et sa traduction en programme C sera comme suit :

```
Algorithme pair_impair
      Variables
            a, r : entier
Début
      Lire (a)
      r ← a mod 2
      si (r = 0) alors
         écrire ('a est pair')
      sinon
         écrire ('a est impair')
      finsi
Fin
```

```
#include <stdio.h>

int main(){
      int a, r;
      printf("Donnez une valeur entière : ");
      scanf ("%d" , &a);
      r = a % 2;
      if (r == 0)
            printf("a est pair ");
      else
            printf("a est impair ");
}
```

Glossaire

C

Langage général de programmation orientée objets, très populaire, inventé en 1960 par Dennis Ritchie pour AT&T

Références

https://elearning.univ-bejaia.dz/course/view.php?id=14316
https://elearning.univ-bejaia.dz/course/view.php?id=14316

Bibliographie

- ***CORMEN, T. (2013). Algorithmes : Notions de base. Informatique. Editions***
- ***DUNOD. GAUDEL, M., Soria, M., and Froidevaux, C. (1987). Types de données et algorithmes. Number vol. 1 in Collection didactique. Institut national de recherche en informatique et en automatique.***