Institut Supérieur d'informatique et de Mathématiques de Monastir

Algorithmique et Structures de Données

Akil ELKAMEL

akil.elkamel@isimm.rnu.tn

Chapitre 1:

Introduction à l'algorithmique

I. Introduction

Qu'est-ce un algorithme?

- Un algorithme est la description d'une méthode pour effectuer une tâche précise : l'algorithme prend en compte des données de départ et donne les instructions à effectuer pour parvenir au résultat cherché.

- La manière de formuler les instructions n'a pas d'importance, du moment qu'elles sont claires, non ambiguës et applicables à l'ensemble des données initiales

I. Introduction

Qu'est-ce une structure de données ?

- Une structure de données est un format spécial destiné à **organiser, traiter, extraire** et **stocker** des données.

- Il existe plusieurs types de structures plus ou moins complexes, tous visent à organiser les données pour répondre à un besoin précis, afin de pouvoir y accéder et les traiter de façon appropriée.

I. Introduction

Qu'est-ce une structure de données ?

- En informatique, une structure de données peut être sélectionnée ou conçue pour stocker des données de manière à pouvoir manipuler ces dernières à l'aide de **plusieurs algorithmes**.

- Chaque structure de données contient des informations sur la valeur des données, les relations entre elles et les fonctions applicables.

- Exemple: sur la structure de donnée **tableau d'entiers en** peut appliquer des algorithmes de recherche ou de tri.

I. Introduction

Quel langage choisir?

- Cela dépend à qui l'algorithme est destiné :

➔ Si c'est à une personne on utilisera sa langue habituelle,

➔ Si c'est à un ordinateur on utilisera un langage de programmation particulier pour cette machine.

➢ Ici, on suppose que l'on va traduire les algorithmes en des langages de même type qui seront compréhensibles par différentes machines.

➔ On choisira le langage de programmation C pour traduire les algorithmes et ainsi de pouvoir les faire fonctionner.

II. Historique du langage C

- Le langage C a été conçu en 1972 par Dennis Richie et Ken Thompson chercheurs aux Bell Labs

- Il a été conçu à l'origine pour l'écriture du système d'exploitation UNIX (90-95% du noyau est écrit en C) et s'est vite imposé comme le langage de programmation sous UNIX.

- Il a été normalisé en 1989 par le comité X3J11 de l'American National Standards Institute (ANSI)

III. Caractéristiques du C

- Le langage C est un langage **évolué**, **modulaire** et **structuré**, assez proche du langage machine destiné des applications de contrôle de processus (gestion d'entrées/sorties, application temps réel…).

- Le langage C possède un peu d'instructions. Il fait par contre appel à des bibliothèques.

Exemple :
math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées/sorties standard

III. Caractéristiques du C

Nécessite de la déclaration préalable : normalement, tout objet C doit être déclaré avant d'être utilisé. S'il ne l'est pas, il est considéré comme étant du type entier.

Format libre : la mise en page des divers composants d'un programme est totalement libre. Cette possibilité doit être exploitée pour rendre les programmes lisibles.

Transportable : les entrées/sorties sont réunies dans une librairie externe au langage

Souple et permissivité : peu de vérifications et d'interdits, hormis la syntaxe.

III. Caractéristiques du C

- C est un langage compilé ≠ langage interprété

- **Compiler un programme** c'est traduire un texte (code source) d'un langage de haut niveau (langage C) en code de bas niveau (code machine), de manière à ce que le système d'exploitation puisse, au besoin, déclencher l'exécution de ce programme. Dans un langage compilé, l'étape de traduction a lieu une fois pour toutes.

- **Interpréte un programmer**, c'est faire en même temps la traduction et l'exécution du texte d'un langage de haut niveau (un script). ➔ Dans un langage interprété, l'étape de traduction a lieu à chaque exécution.

III. Caractéristiques du C

- **Avantage**: compilation indépendante de chaque module

Si le code source d'un module est modifié,

Seuls ce module et ceux qui l'utilisent doivent être recompilés

Compilation par GCC: gcc –o Application programme.c

Compilation par Microsoft Visual C++ :

cl programme.c /Application.exe

- **Étapes de la compilation** :
  1. Analyse lexicale
  2. Analyse syntaxique
  3. Analyse sémantique
  4. Génération du code
  5. Édition de liens

III. Caractéristiques du C

1) Analyse lexicale

Identifie les lexèmes (unités lexicales du langage). Les espaces sont inutiles (3*x+1 ou 3 * x + 1), sauf comme séparateurs (int x, intx).

Exemples d'erreur lexicale :
code source int x = @;
compilation error: stray '@' in program
Erreur détectée uniquement au moment de l'analyse sémantique :

code source intx = 0;
compilation error: 'intx' undeclared (first use in this function)

III. Caractéristiques du C

2) Analyse syntaxique

Trouve la structure syntaxique, (arbre syntaxique), et teste l'appartenance au langage.

Exemple : Dans l'expression x = 3 * x + 1, est-ce que la sous-suite x + 1 correspond à une structure syntaxique ?

Exemple d'erreur syntaxique :

code source Un **else** sans **if**

compilation error: expected expression before 'else'

III. Caractéristiques du C

3) Analyse sémantique

Trouver le sens des différentes actions voulues par le programmeur.

- Quels sont les objets manipulés par le programme,
- Quelles sont les propriétés de ces objets,
- Quelles sont les actions du programme sur ces objets.

Beaucoup d'erreurs peuvent apparaître durant cette phase : identificateur utilisé mais non déclaré (la réciproque génère un warning avec l'option −Wall), opération n'ayant aucun sens, etc.

code source variable x utilisée mais non déclarée

compilation error: 'x' undeclared (first use in this function)

III. Caractéristiques du C

4) Génération de code

- La génération du code se décompose en 3 phases:
  - **Le traitement par le pré-processeur:** réalise des transformations d'ordre purement textuel (inclusion d'autres fichiers sources, etc.) (*.i)
  - **la compilation:** traduit le fichier généré par le pré-processeur en langage assembleur (*.s)
  - **l'assemblage:** transforme le code assembleur en fichier binaire appelé fichier *objet (*.o)*

III. Caractéristiques du C

5) Édition des liens

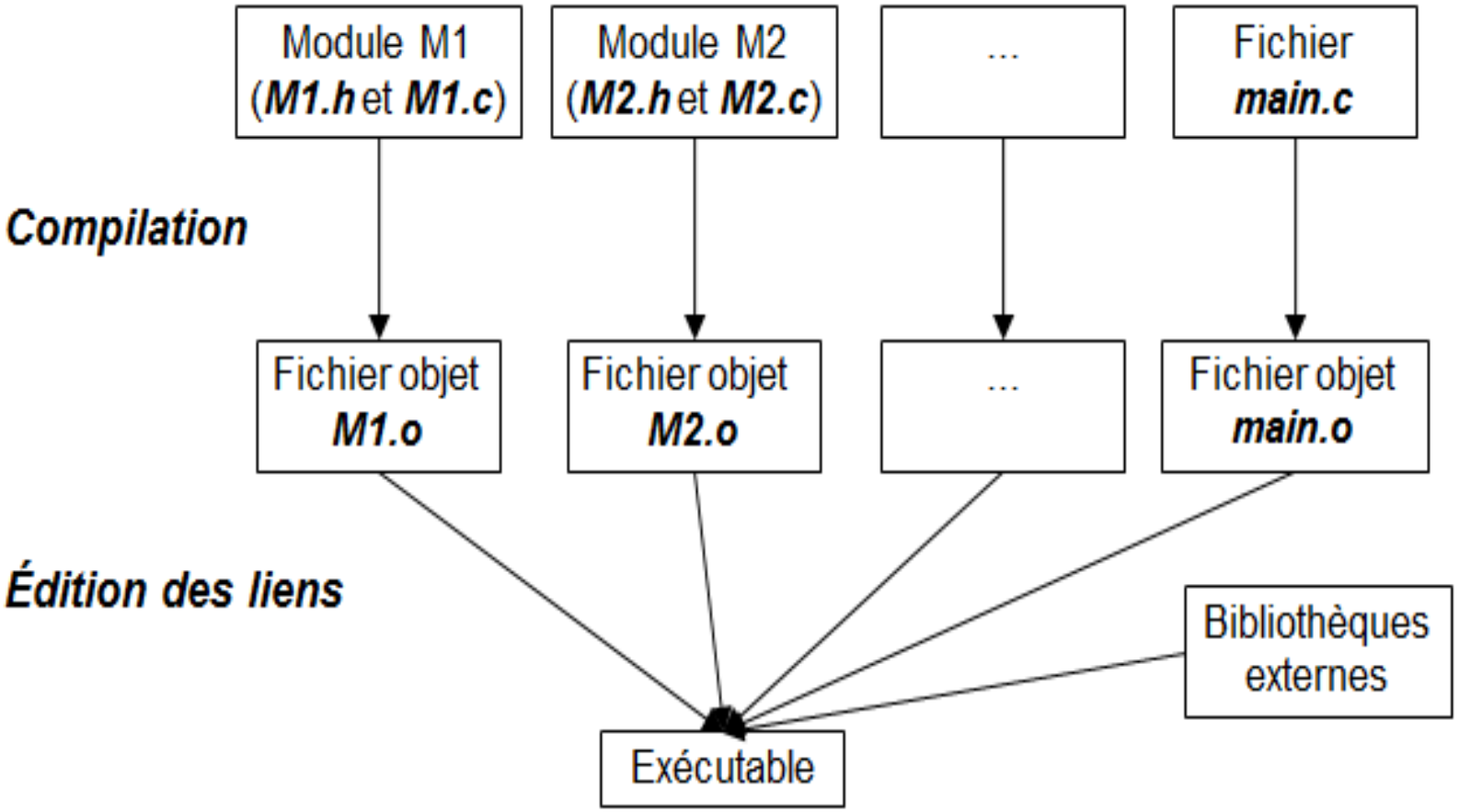
Le code objet des fonctions externes (bibliothèques) est ajouté à l'exécutable.
Le point d'entrée dans le programme est choisi (main).
Insertion de données de débogage (détection et correction des bugs) (option −g).
Exemple: gcc -g -o application programme.c

code source Oublie de stdio.h et utilisation de printf.
compilation warning: incompatible implicit declaration of built-in function 'printf'

code source Pas de fonction principale (main)
compilation Undefined symbols: "_main", ...

III. Caractéristiques du C

5) Édition des liens



IIV. Structure d'un programme C

Pour simplifier les choses, dans ce cours, en considère qu'un programme C est écrit dans un seul fichier composé de :

- ***Un entête*** (header) constitué de méta instructions ou **directives** destinées au préprocesseur. Ces directives permettent d'effectuer des manipulations sur le texte du programme source avant la compilation : Une directive du préprocesseur est une ligne de programme source commençant par le caractère dièse (**#**).
- ***Un bloc principal*** appelé **main ().** Il contient des instructions élémentaires ainsi que les appels aux modules (fonctions) considérés par le compilateur comme des entités autonomes. Chaque appel à une fonction peut être considéré comme une instruction.
- ***Le corps des fonctions*** placées avant ou après la fonction main () dans un ordre quelconque, les unes après les autres.

IV. Structure d'un programme C

```
#include …
#define …

fonction1
fonction2
….

main()
{
    définitions,
    instructions,
    et appel de
    fonctions.
}

fonction1 {…}
fonction2 {…}
…..
```

Entete

Prototype des fonctions

Bloc principal

Définition des fonctions

```
#include …
#define …

fonction1 {…}
fonction2 {…}
…..

main()
{
    définitions,
    instructions,
    et appel de
    fonctions.
}
```

Entete

Définition des fonctions

Bloc principal

IV. Structure d'un programme C

Remarques :

Il est préférable de **commenter** les instructions difficiles.

- Ceci permet d'expliquer la signification de ces dernières au lecteur même s'il ne connaît pas le langage de programmation.
- Les commentaires en C peuvent être placés à n'importe quel endroit du programme.
- Ils commencent par /* et se terminent par */.
- Le langage C distingue les minuscules, des majuscules. Les mots réservés du langage C doivent être écrits en minuscules.

V. Les composantes élémentaires d'un programme C

1) Identificateurs et mots-clés

- Un identificateur est le nom donné à une variable, une constante, une fonction, etc.

- Ce nom est composé de lettres non accentuées, de chiffres et du caractère dans un ordre quelconque sauf pour le premier caractère qui ne peut pas être un chiffre.

- Rq. Certains mots ne peuvent pas être utilisés comme identificateurs car ils sont réservés.

V. Les composantes élémentaires d'un programme C

1) Identificateurs et mots-clés

- Les mots clés ne vous diront rien pour le moment, tout ce qu'il faut savoir c'est que vous ne devez pas utiliser ces mots pour nommer vos variables et fonctions :

auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.

V. Les composantes élémentaires d'un programme C

2) Variables

- Une variable est un espace mémoire que le programme réserve lors de son exécution au moment de sa déclaration.
- C'est une donnée dont la valeur est modifiable.
- Dans tous les langages, une définition de variable a les rôles suivants :

1. définir le domaine de valeur de cette variable (taille en mémoire et représentation machine) ;
2. définir les opérations possibles sur cette variable ;
3. définir le domaine de visibilité de cette variable ;
4. permettre à l'environnement d'exécution du programme d'associer le nom de la variable à une adresse mémoire ;

V. Les composantes élémentaires d'un programme C

2) Variables

a) Déclaration de variable

Lorsqu'on utilise un langage de programmation **déclaratif**, il faut toujours donner le type des variables avant de les utiliser. C'est ce qu'on appelle la déclaration.

Syntaxe :

nom_type nomVariable ;

nom_type nomVariable_1, nomVariable_2, nomVariable_N ;

Exemple :

```
int A ; /*déclaration d'une variable de type entier*/
float X, Y ; /*déclaration de deux variables de type réel*/
```

V. Les composantes élémentaires d'un programme C

2) Variables

a) Affectation de variable

Le symbole ← que nous utilisions en algorithmique correspond au signe = du langage C.

Syntaxe :

```
nom_variable = valeur;
```

Exemple :

```
int x, y ;
x=8 ;
y=x ;
```

Une variable peut être initialisée lors de sa déclaration.

Exemple :

```
int a=3 ;
float c= 3.2;
```

V. Les composantes élémentaires d'un programme C

3) Constantes

Une constante est un identificateur qui contient une valeur qui ne sera jamais modifiée au cours du programme.

Par exemple, la constante PI peut être fixée à 3.14.
Les constantes peuvent être :

- des nombres entiers (12),
- des nombres réels (20.6),
- un caractère ('a')
- une chaîne de caractères ("bonjour").

Par conséquence on les écrit souvent en majuscules pour les différencier des autres identificateurs.

V. Les composantes élémentaires d'un programme C

3) Constantes

Déclaration de constante

Il existe deux façons pour déclarer une constante :

Syntaxe 1 :

```
#define nom_constante valeur_constante
```

- Cette déclaration est effectuée dans la partie pré-processeurs (après l'inclusion des bibliothèques)
- Elle permet la déclaration d'une constante qui sera visible par tout le programme et sa durée de vie est celle de toute l'application.

Exemples :

```
#define  PI  3.14    /* Rq sans ;  */
#define TVA  0.16
```

V. Les composantes élémentaires d'un programme C

3) Constantes

Syntaxe 2 :

```
const type nom_constante = valeur_constante ;
```

- Cette déclaration est effectuée dans le bloc
- Elle permet la déclaration d'une constante qui sera visible seulement dans le bloc et elle sera détruite en sortant du bloc.

Exemple :

```
const float PI = 3.14 ;      /* Rq avec ; */
```

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

- C est un langage **typé** : toute variable, constante ou fonction est dotée d'un type précis.
- Les types **de bases** de C sont au nombre de six :
  - void
  - int
  - char
  - float
  - double
  - long double

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

void

- C'est le type vide. Il a été introduit par la norme ANSI.

- Il est surtout utilisé pour préciser les fonctions sans argument ou sans retour.

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

int

- C'est le type entier. Ce type se décline avec des qualificatifs pour préciser sa taille (**long** ou **short**), et le fait qu'il soit uniquement positif (**unsigned**) ou positif et négatif (**signed**). Le qualificatif **signed** est appliqué par défaut, ainsi il n'y a pas de différence entre une variable de type **int** et une variable de type **signed int**.

Exemples

```
long int a ;    /*     long a ; */
short int b ;   /*    short b ; */
unsigned int c ;
signed int d ;    /* int d ;  */
```

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

char

- Ce type est très proche de l'octet. Il représente un entier sur huit bits.
- Sa valeur peut évoluer entre -128 et +127.
- Il est le support des caractères au sens commun du terme.
- Ces caractères sont représentés par la table ASCII.
- Comme le type **int** le type **char** peut être qualifié de manière à être signé ou non.

Exemple

```
char c1; signed char c2; unsigned char c3;
```

Rq. La norme ANSI introduit un type permettant de supporter des alphabets comprenant plus de 255 signes, ce type est appelé **wchar_t**. Il est défini dans le fichier <**stddef.h**>.

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

float

- Ce type sert pour les calculs avec des parties décimales.

double

- C'est un type qui permet de représenter des valeurs ayant une partie décimale avec une plus grande précision que le type **float**. Comme nous le verrons dans l'expression des constantes et dans les calculs, ce type est le plus courant pour représenter des valeurs avec parties décimales.

long double

- Ce type est récent, il permet de représenter des nombres avec parties décimales qui nécessitent une très grande précision.

V. Les composantes élémentaires d'un programme C

4) Types prédéfinis

Type Année	PDP 11 1970	Intel 486 1989	Sparc 1993	Pentium 1993	Alpha 1994
char	8 bits	8bits	8bits	8bits	8bits
short	16 bits	16 bits	16 bits	16 bits	16 bits
int	16 bits	16 bits	32 bits	32 bits	32 bits
long	32 bits	32 bits	32 bits	32 bits	64 bits
float	32 bits	32 bits	32 bits	32 bits	32 bits
double	64 bits	64 bits	64 bits	64 bits	64 bits
long double	64 bits	64 bits	64 bits	64 bits	128 bits

V. Les composantes élémentaires d'un programme C

5) Déclaration de nouveaux types

Le langage C permet de définir de nouveaux types de variables en utilisant le mot clé **typedef** selon la syntaxe:

typedef type_de_base nom_nouveau_type ;

Exemple

```
#define  PI 3.14
typedef float reel ;
void main() {
        reel perimetre, rayon =8,7;
        perimetre = 2* PI* rayon;
        …
}
```