Programmation C

Structure d'un programme, Compilation

ING1

CY Tech

Structure d'un programme

Préambule

- Plusieurs approches de programmation (parmi les languages de 3ème génération)
  - fonctionnelle : appel de fonction mathématiques (Caml, Scheme)
  - procédurale : suite d'instructions ou d'appel de fonctions $\Rightarrow$ automate (C, Pascal, C++)
  - logique : utilisation de formule logique ; utilisé en IA (Prolog)
  - objet : ensemble d'objets communiquant ensemble (Java, C++)
- L'algorithmique est souvent plus intuitive pour la programmation procédurale
- Type abstrait : approche programmation objet

Algorithmes et Programmes

Algorithme

- Précis
  - Comportement décrit à partir d'actions élémentaires
- Complet
  - Tous les cas de figures doivent être pris en compte
- Non ambigu
  - Face à une situation : un seul comportement

Algorithmes et Programmes

- L'écriture d'un algorithme est une étape de la conception de programme

- La programmation est une autre étape de la conception de programme

Définitions

- Un algorithme est une séquence finie d'actions, dont le déroulement suit un ordre prévu à l'avance.
- Un programme est la traduction (codage) d'un algorithme dans un langage de programmation afin de pouvoir être exécuté sur un ordinateur.

Processus de programmation en C

- Écriture de code source (traduction de l'algorithme) dans un fichier portant comme extension : `.c`
- Compilation du code source : traduction du code source vers un fichier exécutable par la machine
- Exécution du fichier généré

Programme C

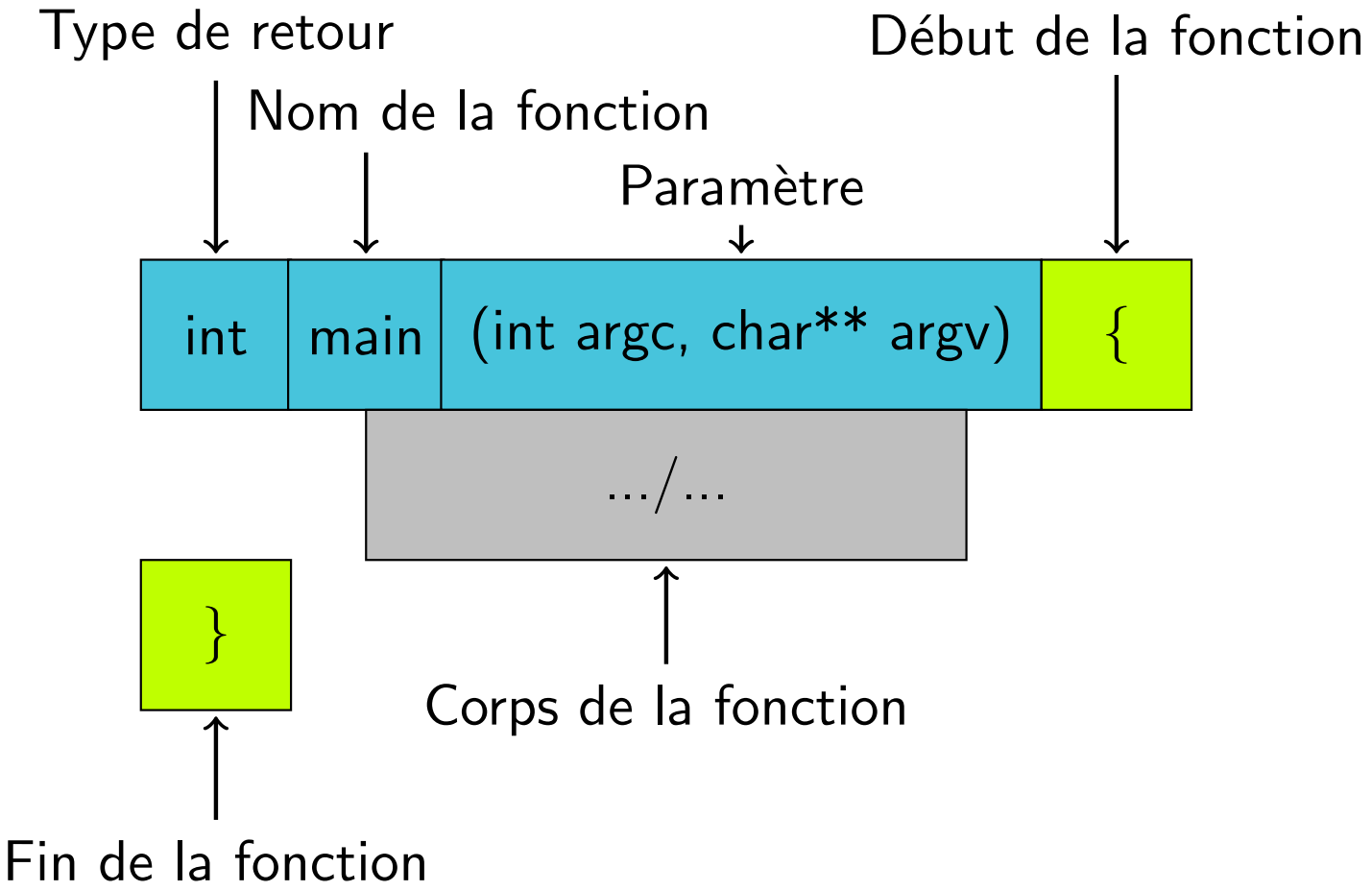
- Ensemble de fonctions
- Une seule doit s'appeler : `main`
- Exécution d'un programme : exécution de *la* fonction `main`

Fonctions

Description

- Nom de la fonction
- Paramètres
- Corps
- Type de résultat

Fonctions


Type de retour
Nom de la fonction
Paramètre
Début de la fonction
int
main
(int argc, char** argv)
{
.../...
}
Corps de la fonction
Fin de la fonction

Algorithme minimum

Affichage d'un message

programme Hello
  écrire "Hello World !"
fin programme

Programme minimum

Affichage d'un message (en C) : `hello_world.c`

```
int main (void) {
    printf("Hello World !");
    return (0);
}
```

- À taper dans un éditeur de texte
- Nécessite une compilation
- Puis doit être exécuté
- Affiche : `Hello World !` sur la ligne de commande

Remarques

- Programme syntaxiquement correct
- Mauvais programme
  - Pas de commentaire de programme
  - Mauvaise déclaration du main
  - Que fait le code ?
  - ...

Programme minimum correct

Affichage d'un message (en C) : hello_world.c

```c
/*! Commentaire de fichier (doxygen) */

/* Inclusion des entetes de librairies */
#include <stdio.h>

/*! Commentaire de la fonction main (doxygen) */
int main (int argc, char** argv) {
    // Affichage d'un message
    printf("Hello World !");
    // Fin du programme : tout est OK.
    return (0);
}
```

Commentaires

Types de commentaires

- Commentaires `C`
- Commentaires `doxygen`

Rôle, utilité, justification

- Annote le code que vous faites
- Permet de comprendre rapidement le code
- Permet une vérification plus rapide
- Peut permettre de générer une documentation technique
- Vous oblige à une rigueur
- Fait gagner du temps

Commentaires C

Notation

- // : Indique un commentaire jusqu'à la fin de la ligne (mais n'est pas accepté par les normes ANSI [a])
- /* : Indique un début de bloc de commentaires (une ou plusieurs lignes)
- */ : Indique une fin de bloc de commentaires
- Attention à ne pas imbriquer plusieurs blocs de commentaires

a. American National Standards Institute

- Se positionne juste avant le code (ou bloc de code) à commenter
- Doit rester dans la même langue

Commentaires doxygen

- Permet de réaliser une documentation automatiquement
- Commentaires de fichier : décrit l'utilité du fichier
- Commentaires de fonction : décrit l'utilité de la fonction, ses paramètres, ses valeurs de retour, pré-condition, post-condition, ...
- Plus beaucoup d'autres !
  - `http://www.doxygen.nl/manual/`

Commentaires doxygen

Notation

- /*! ou /** : Indique un début de bloc de commentaires doxygen
- */ : Indique une fin de bloc de commentaires
- Utilisation de balises :
  - `\version num description` : version en cours, ainsi qu'une brève description
  - `\author nom` : qui a écrit le fichier ou la fonction (une balise par auteur)
  - `\date date` : date de création/modification
  - `\brief description` : description courte
  - `Description longue`
  - `\remarks description` : éventuellement des remarques générales

Commentaires doxygen

Notation

- Utilisation de balises pour les commentaires de fichiers :
  - `\file nom_fichier` : indique le fichier qui est concerné
- Utilisation de balises pour les commentaires de fonctions :
  - `\fn prototype_fonction` : indique la fonction qui est concernée (type de retour, nom de la fonction et les paramètres)
  - `\param nom_parametre` : description du paramètre (autant de balises que de paramètres)
  - `\return description` : description du retour de la fonction (balise non présente le commentaire est pour une procédure)

Commentaires doxygen

Commentaire de fichier (fichier tp1.c)

```
/*!
\file tp1.c
\author Elisabeth Ranisavljevic <erc@eisti.eu>
\version 0.1 Premier jet
\date 3 septembre 2017
\brief Premier contact avec la programmation C
*/
```

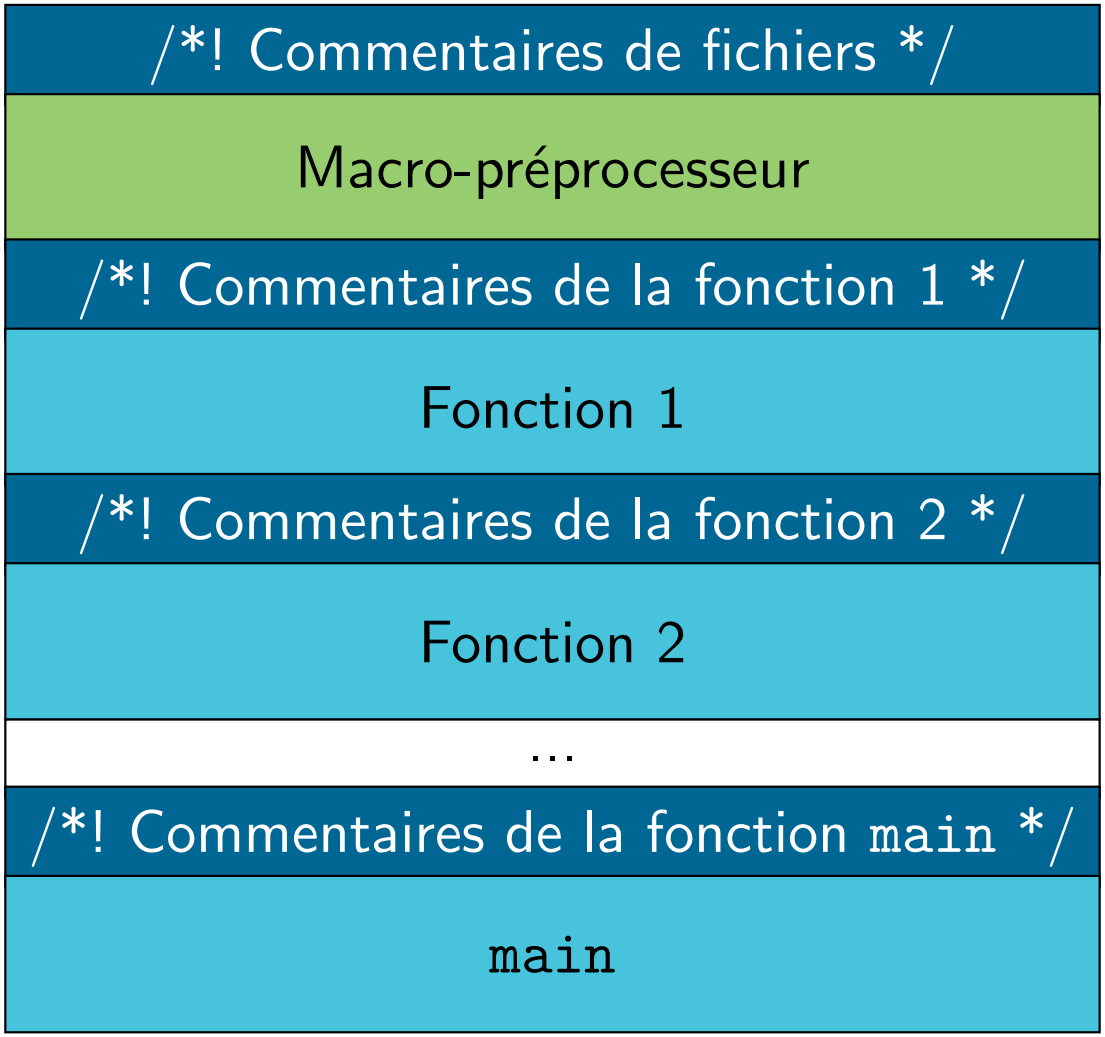
Commentaires doxygen

Commentaire de fonction (la fonction `main`)

```
/*!
\fn int main (int argc, char** argv)
\author Elisabeth Ranisavljevic <erc@eisti.eu>
\version 0.1 Premier jet
\date 3 septembre 2017
\brief Hello world !
\param argc nombre d'arguments en entree
\param argv valeur des arguments en entree
\return 0 si tout c'est bien passe
*/
int main (int argc, char** argv) {
   .../...
   return (0);
}
```

Aspect général

Structure d'un fichier `.c`



Types de données

Type de données basique

- entier : `char, int`
- réel :`float, double`
- caractère : `char`
- Chaque type possède un codage différent sur machine

Extension des types

Deux notions orthogonales :

- Taille : `short`, `long`
  - Uniquement pour les `int` et `double`
- Signe : `unsigned, signed` par défaut
  - Uniquement pour les `char, int, float`

Constantes

Constantes

- Invariable pendant l'exécution
- Possède un type
- Constantes numériques
  - 14, 3.14, Oxbfabcdd0, ...
- Constantes de type caractère
  - 'A', 'z', '1', ...
- Séquences d'échappement
  - Caractère provoquant un "affichage" particulier
  - \n, \t, \v, \b, \r, \f, \', \", \\, *nnn*, \x*nnn*, ...
- Constantes de type chaîne de caractères
  - "Coucou", "Aujourd\'hui", "Hello\nWorld !", "", ...

Table ASCII

American Standard Code for Information Interchange

- Codage de 128 caractères
- Equivalence entre un nombre (en base 2, 8, 10 ou 16) et un caractère
- Exemples :

Nb	Char	Signification
13	\n	Retour chariot
33	!	Point d'exclamation
49	1	Chiffre un
65	A	Lettre latine capitale A
97	a	Lettre latine minuscule a

Variables

Variables

- Valeur peut changer
- Nécessite un nom et un type
- Déclaration : `type nom ;`
- Règles de programmation : commentaires de variables et nom explicite !
- Pas d'accent, ni ponctuation
  - Seul les caractères alphanumériques et le _
- Commence obligatoirement par une lettre
- Affectation $\Leftrightarrow =$

Variables

Exemple à ne pas faire

```
int i;      int a, b, c;
```

Exemple à faire

- `int i; // Variable de boucle`
- `int int_coeff_a; // Premier terme de l'équation`
- `int_coeff_a = 5; /* Initialisation du premier terme de l'équation */`
- `char_reponse = 'o';`
- `str_reponse = "oui"; /* Ne fonctionne pas */`

Compilation

Compilation

- Convertir un fichier texte en un fichier binaire, exécutable.
- Commande : `gcc [options] fichiers`

```
gcc tp.c
```

- Exécution (par défaut, l'exécutable s'appelle `a.out`)

```
./a.out
```

- Options `gcc` :
  - `-Wall` : Affiche tous les *warnings* par le compilateur
  - `-Werror` : Transforme tous les *warnings* en erreurs
  - `-o nomFichier` : Précise le nom du fichier en sortie
  - Pour plus d'options, voir le manuel de `gcc`

```
gcc -Wall tp.c -o monProgramme
```

Compilation

4 phases

1. Pré-processing : passage au pré-processeur
2. Compilation : compilation en langage assembleur
3. Assemblage : conversion du langage assembleur en code machine
4. Édition des liens : liens vers les bibliothèques

Compilation

Pré-processing

```
gcc -E toto.c -o toto.i
```

- Transformation textuelle uniquement : opération de substitution
  - supression des commentaires
  - inclusion des fichiers `.h` dans le fichier `.c` : `#include`
  - traitement des directives de compilation qui commencent par un # : *define, ifndef, endif, include*, ...
- `toto.i` est un fichier texte en langage C, lisible
- Contrôle la **syntaxe** du programme

Compilation

Compilation

```
gcc -S toto.i -o toto.s
```

- Le code C est transformé en assembleur
- `toto.s` est un fichier texte lisible, mais difficilement compréhensible

Compilation

Assemblage

```
gcc -c toto.s -o toto.o
```

- Le code assembleur est transformé en code machine binaire
- `toto.o` est un fichier binaire

Compilation

Édition des liens

```
gcc toto.o -o toto
```

- `toto.o` est incomplet : il ne contient pas le code les fonctions inclues
  - `#include<...>` n'inclue que le **prototype** des fonctions, pas leurs codes
- Réunir le fichier *objet* (`toto.o`) et les fonctions contenues dans les bibliothèques pour produire un programme exécutable

Doxygen

Documentation doxygen

- Nécessite un fichier de configuration : *Doxyfile* ou *doxyfile*
  - peut se générer avec la commande `doxygen`

```
doxygen -g
```

- Configuration adaptée par rapport au cas d'utilisation
  - `PROJECT_NAME`, `PROJECT_NUMBER`, `OUTPUT_DIRECTORY`, ...
- Compilation de la documentation
  - génère ce qui a été demandé : LaTeX, HTML, ...

```
doxygen Doxyfile
```