

Programmation en C

Généralités

Alain CROUZIL

`alain.crouzil@irit.fr`

Département d'Informatique (Ddl)

Institut de Recherche en Informatique de Toulouse (IRIT)
Équipe Computational Imaging and Vision (MINDS)

Faculté Sciences et Ingénierie (FSI)
Université Toulouse III – Paul Sabatier (UPS)

Licence Informatique – Licence MIASHS
2019-2020

Sommaire

- 1 Motivations
- 2 Fonctionnement de cette UE
- 3 Exemples de programmes simples
- 4 Chaîne de production d'un programme

Sommaire

1 Motivations

- Historique
- Caractéristiques du langage
- Prérequis pour la suite
- Objectifs

2 Fonctionnement de cette UE

3 Exemples de programmes simples

4 Chaîne de production d'un programme

Historique

Création

- Denis RITCHIE et Ken THOMPSON (Laboratoires Bell), 1972
- Écriture du système d'exploitation UNIX
- Première définition (K&R) : KERNIGHAN et RITCHIE, *The C programming language*, 1978

Normalisations

Organismes :

- ANSI : *American National Standard Institute*
- ISO : *International Standardization Organization*
- IEC : *International Electrotechnical Commission*

Normes :

- | | |
|-------------------------------|-----------------------|
| • C89 : ANSI, 1989 (C ANSI) | • C99 : ISO/IEC, 1999 |
| • C90 : ISO/IEC, 1990 (C ISO) | • C11 : ISO/IEC, 2011 |

Caractéristiques du langage

Langage de bas niveau

Le langage C est :

- de bas niveau : efficacité, contrôle
- mais portable

Base d'autres langages

Syntaxe à la base de nombreux langages : C++, Objective-C, C#, Java, JavaScript, PHP...

Souplesse du langage

- Peu de vérifications effectuées par le compilateur
- Problèmes de lisibilité
- Problèmes de mise au point
- Nécessité d'une programmation rigoureuse

Prérequis pour la suite

Exemples en deuxième année de licence

- L2 Informatique
 - Algorithmique et programmation
 - Projet S3
 - Systèmes 2
 - Structures de données
- L2 MIASHS
 - Fondements de la programmation
 - Systèmes 2
 - Projet collaboratif et transversal

Objectifs

Compétences visées

- Identifier dans des programmes des erreurs de syntaxe et d'exécution
- Analyser le comportement de programmes simples
- Compiler, tester et mettre au point des programmes simples
- Implémenter en langage C des algorithmes simples

Points importants

Maîtriser :

- la syntaxe du langage C
- le concept de pointeur
- le mécanisme du passage des paramètres

Plutôt que de couvrir tous les aspects du langage, nous préférons que vous puissiez en acquérir les bases.

Sommaire

1 Motivations

2 Fonctionnement de cette UE

- Organisation
- Ressources
- Évaluation
- QCM en cours

3 Exemples de programmes simples

4 Chaîne de production d'un programme

Organisation

Séances

- 12 heures de cours : 6 séances de 2 heures
- 16 heures de TP : 8 séances de 2 heures

Conséquences

- Travail personnel nécessaire
- Rentabiliser au maximum les cours

Ressources

En séance

- Exemples
- Explications
- Conseils

En dehors des séances

- Diaporamas utilisés en cours
- Support complémentaire
- Sujets des TP
- Autres ressources



Le cours ne sera pas refait en séance de TP.

Modalités de contrôle des connaissances et des aptitudes

Modalités officielles

● Session 1

- Contrôle continu (CC) : 25%
- Contrôle terminal (CT) : 75% (2 heures)

● Session 2

- Report CC
- CT : 75% (2 heures)

Modalités du contrôle continu

Évaluations sur Moodle d'une durée de 20 à 30 minutes, prévues lors des séances de TP 4, 6 et 8.

À vos boîtiers !

1



Sommaire

- 1 Motivations
- 2 Fonctionnement de cette UE
- 3 Exemples de programmes simples**
- 4 Chaîne de production d'un programme

Exemples de programmes (1/2)

bonjour.c

```
1 #include <stdio.h> /* Pour pouvoir faire des entrées-sorties */
2
3 int main(void) /* En-tête de la fonction principale */
4 {
5     printf("Bonjour_\n"); /* Affichage sur la sortie standard (l'écran, par défaut) */
6
7     return 0; /* Envoi d'un code de retour au système d'exploitation */
8 }
```

Exemples de programmes (2/2)

exemple.c

```
1  /* Conversion miles -> km */
2  #include <stdio.h>
3
4  #define COEFFICIENT 1.609344
5
6  int main(void)
7  {
8      double miles,km;
9
10     printf("Donnez la distance en miles: ");
11     scanf("%lf",&miles);
12     km=COEFFICIENT*miles;
13     printf("%f_mile(s) _=%f_km\n",miles,km);
14
15     return 0;
16 }
```

Sommaire

- 1 Motivations
- 2 Fonctionnement de cette UE
- 3 Exemples de programmes simples
- 4 Chaîne de production d'un programme
 - Systèmes de traduction
 - Outils du programmeur C
 - Exemple
 - Étapes
 - Différents types d'erreurs

Systèmes de traduction (1/5)

Traduction

La machine ne sait traiter que :

- des instructions simples codées en binaire ;
- des données codées en binaire.

⇒ Nécessité de traduire les programmes en langage machine.

Deux stratégies principales

- interprétation
- compilation + exécution

Systèmes de traduction (2/5)

Interprétation



- Chaque ligne du programme source est analysée par l'interpréteur. Si elle respecte la syntaxe du langage, il la traduit en langage machine et l'exécute.
- L'interpréteur doit être utilisé à chaque fois que l'on veut exécuter le programme.
- Exemples de langage interprétés : sh, bash, Perl, PHP, JavaScript, ...

Compilation + exécution



- Le compilateur vérifie la syntaxe du programme source et le traduit en totalité en un programme exécutable stocké dans un fichier exécutable.
- Le programme peut être exécuté indépendamment du compilateur.
- Exemples de langages compilés : Pascal, C, C++...

Systèmes de traduction (3/5)

Interprétation : avantages et inconvénients

Avantages :

- permet de tester immédiatement des petites parties d'un programme sans devoir le re-compiler dans son ensemble ;
- permet de produire des programmes portables : nécessite le source et l'interpréteur.

Inconvénients :

- exécution plus lente car l'interpréteur doit analyser et traduire chaque ligne du programme source pour l'exécuter ;
- nécessité de disposer du logiciel interpréteur pour pouvoir exécuter un programme.

Systèmes de traduction (4/5)

Compilation : avantages et inconvénients

Avantages :

- exécution plus rapide ;
- programme exécutable autonome (pas besoin du logiciel compilateur pour pouvoir l'exécuter).

Inconvénients :

- tester l'effet d'une petite modification nécessite de compiler à nouveau tout le programme ;
- la compilation d'un gros programme peut prendre du temps ;
- le programme exécutable n'est pas complètement portable : par exemple, il faut relancer une compilation si on change de système d'exploitation.

Systèmes de traduction (5/5)

Systèmes hybrides



- L'interprétation du pseudo-code (« *bytecode* ») est plus rapide que celle du programme source.
- Le bytecode est portable : il suffit d'avoir un interpréteur de bytecode.

Outils du programmeur C

Deux outils de base

- Éditeur de texte : `nedit`, `Kate`, `TextWrangler`, `Atom`...
- Compilateur : `gcc` (Gnu C Compiler), `LLVM/Clang`...

Outils avancés

- Débogueur : `gdb`, `LLDB`...
- Environnement de développement intégré (« *Integrated Development Environment* ») : `Code::Blocks`, `Eclipse CDT`, `Xcode`...
- Gestionnaire des dépendances : `make`, `CMake`...
- Outil d'analyse statique : `lint`, `Frama-C`, `Clang Static Analyzer`...
- Outil d'analyse dynamique : `Valgrind`...

Démo



Compilation avec `gcc` et exécution d'un programme simple sous Unix

Compilation

```
gcc_exemple.c_o_exemple
```

Exécution

```
./exemple
```

Compilation avec vérification plus stricte (fortement conseillée)

```
gcc_Wall_exemple.c_o_exemple
```


Étapes de la compilation

exemple.c



COMPILATION (`gcc exemple.c -o exemple`)

Compilation sans édition de liens (`gcc -c exemple.c`)

- Préprocesseur
- Vérification syntaxique
- Production du code objet



exemple.o



Édition de liens (`gcc exemple.o -o exemple`)

- Production du code exécutable



exemple

Étapes pour le programmeur

Produire le fichier exécutable

- Compiler le programme (`gcc exemple.c -o exemple`)
- *Tant qu'il reste des erreurs à la compilation*
 - Corriger le programme source (*éditer exemple.c et sauver*)
 - Compiler le programme (`gcc exemple.c -o exemple`)

Mettre au point le programme

- Écrire le programme source (*créer exemple.c avec un éditeur de texte*)
- Produire le fichier exécutable (*voir ci-dessus*)
- Exécuter le programme (`./exemple`)
- *Tant qu'il reste des erreurs d'exécution ou de conception*
 - Corriger le programme source (*éditer exemple.c et sauver*)
 - Produire le fichier exécutable (*voir ci-dessus*)
 - Exécuter le programme (`./exemple`)

Différents types d'erreurs (1/2)

Erreurs de syntaxe

- Elles sont détectées par le compilateur.

Erreurs à l'exécution

- Elles se manifestent par un arrêt imprévu de l'exécution avec un message affiché par le système d'exploitation.
- Elles peuvent être provoquées par une instruction du programme qui demande quelque chose d'impossible ou d'interdit (division par 0, accès à une zone mémoire inexistante ou protégée...).
- La syntaxe du langage C étant souple, des fautes de frappes n'entraînent pas une erreur de syntaxe et ne sont pas détectées par le compilateur.
- Elles sont parfois difficiles à corriger (localisation de l'instruction erronée).
- Elles dépendent des données manipulées.
- Elles sont difficiles à éliminer complètement (nécessité de tests mais aussi de techniques de vérification de programmes).

Différents types d'erreurs (2/2)

Erreurs de conception

- Elles sont aussi appelées erreurs sémantiques ou erreurs de logique.
- L'exécution se déroule sans problème, mais le résultat n'est pas celui qui était attendu.
- Elles peuvent provenir d'erreurs de conception de l'algorithme.
- Elles peuvent venir d'erreurs de transcription de l'algorithme dans le langage de programmation (mauvaise connaissance du langage, donc de l'effet produit par chaque instruction).
- Parfois difficiles à corriger.