Formation

Langage C Initiation

Session sur 2 journées

Support de Cours avec Travaux Dirigés

Objet : Support de Cours avec Travaux Dirigés

Votre formateur :
Lionel BEAUDOIN - Tél. : +33 4 28 28 01 40 – contact@com-hightech.com
COM-HIGHTECH – 1 rue de la Mairie - 69440 SAINT ANDRE LA COTE - FRANCE

Table des Matières

1. PRÉSENTATIONS ET INTRODUCTION....3

2. LA STRUCTURE DE BASE D'UN ORDINATEUR....3

2.1. Structure de Base d'un Ordinateur....3

3. LA STRUCTURE D'UN MICROPROCESSEUR....5

3.1. Composants Principaux....5

l'organisation interne des composants :....5

3.2. Principales Instructions....5

3.3. Séquencement des Instructions....5

3.4. Ordonnancement des Instructions....6

3.5. Codage des Instructions....7

4. INTRODUCTION AU LANGAGE C....8

4.1. Historique du langage C....8

4.2. Caractéristiques Générales....9

4.3. Qualités et défauts....10

4.4. Aperçu de la Syntaxe....11

4.5. Brièveté de la Syntaxe....12

4.6. Langage d'Expressions....13

5. DES SOURCES À L'EXÉCUTABLE....14

5.1. Sources....14

5.2. Génération d'un Exécutable....14

5.3. Précompilation....15

5.4. Compilation....15

5.5. Assemblage....16

5.6. Edition des Liens....16

6. ELÉMENTS DU LANGAGE....17

6.1. Eléments lexicaux....17

6.2. Mots clés....17

6.3. Instructions du préprocesseur....17

6.4. Types....17

6.5. Magnitude des Types....18

6.5.1. Types entiers....18

6.5.2. Types à virgule flottante....18

6.5.3. Types élaborés....19

6.5.4. Type booléen....19

6.5.5. Type void....19

6.5.6. Structures....20

6.6. Commentaires....23

6.7. Formatage des fonctions printf et annexes....24

6.8. Structures de contrôle....24

6.9. Fonctions....25

6.10. Prototype....25

6.11. Bibliothèques logicielles....26

6.11.1. La bibliothèque standard....26

6.11.2. Les bibliothèques externes....26

6.12. Allocation mémoire....27

1. Présentations et Introduction

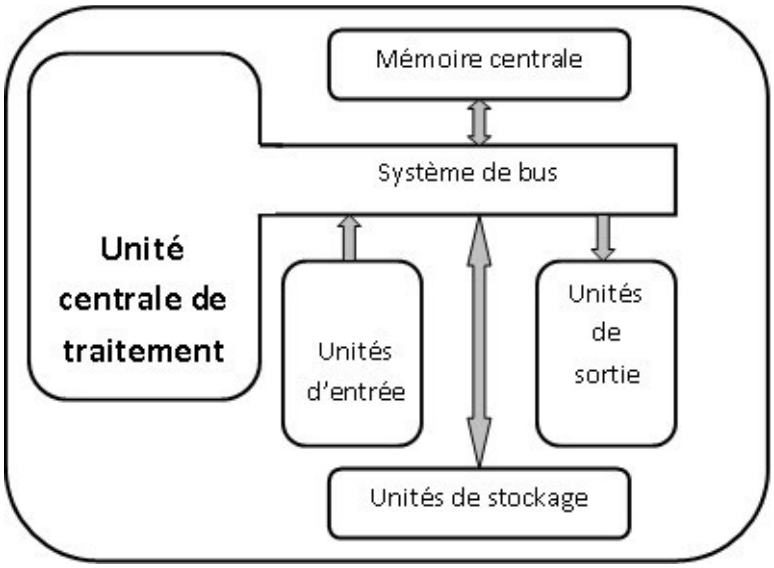
- Présentation des apprenants
- Présentation du formateur
- Pourquoi les chapitres sur la structure d'un ordinateur et d'un micro-processeur ?

2. La Structure de Base d'un Ordinateur

- Source : https://sites.google.com/site/bouzidiinfo/algorithmique-et-programmation/la-structure-de-base-d-un-ordinateur

2.1. Structure de Base d'un Ordinateur

- La structure de base d'un ordinateur comprend les éléments fondamentaux suivants :
  - Une unité centrale de traitement (unité centrale)
  - Une unité de mémoire, appelée mémoire centrale
  - Des unités de stockage
  - Des unités d'entrée/sortie
  - Un système de bus permettant de véhiculer l'information entre l'unité centrale et les autres unités



- Codage de l'information :
  - Les différents constituants de l'ordinateur sont composés de circuits électroniques ou d'éléments magnétiques
  - Ils fonctionnent donc avec des impulsions électriques ou font intervenir un champ magnétique, et donnent lieu à deux états (ouvert ou fermé)
  - L'information élémentaire est le 'bit' et ne peut prendre que deux valeurs notées 0 et 1
  - Généralement, l'ordinateur manipule l'information sous forme de groupement de bits, un groupement de huit bits est nommé 'octet'
  - Avec 1 bit, on peut représenter 2 informations possibles : 0 ou 1
  - Avec 2 bits, on peut représenter 4 informations possibles : 00, 01, 10, 11
  - Avec 3 bits, on peut représenter 8 informations possibles : 000, 001, 010, 011, 100, 101, 110, 111

- Une unité centrale de traitement :
    - L'unité centrale de traitement ou CPU (Central Processing Unit) est le centre de calcul et de contrôle d'un ordinateur, elle constitue le « cerveau » de l'ordinateur
    - Elle réalise toutes les opérations demandées, elle est matérialisée physiquement par un circuit électronique appelé « microprocesseur »

- Une mémoire centrale :
    - La mémoire centrale est un organe de l'ordinateur permettant d'enregistrer, de stocker et de restituer les informations
    - La mémoire centrale se présente sous la forme de petites barrettes que l'on peut enficher dans des supports, appelés Slots
    - La mémoire centrale d'un ordinateur est séparée en deux sections : la mémoire vive et la mémoire morte
    - La mémoire vive (RAM Random Access Memory) :
        - C'est une mémoire où on peut lire et écrire à volonté
        - Cette mémoire est dite « volatile » c'est-à-dire qu'elle perd son contenu dés qu'elle est hors tension
        - La mémoire vive contient en plus des programmes servant à la gestion de l'ordinateur, le programme relatif à un traitement spécifique ainsi que les données qu'il requiert et les résultats qu'il génère
    - La mémoire morte (ROM Read Only Memory) :
        - Elle destinée uniquement à être lue
        - En revanche, son contenu n'est pas altéré par une coupure de courant, c'est une mémoire « rémanente »
        - La mémoire morte, programmée par le constructeur, est essentiellement utilisée pour contenir les logiciels de base servant au démarrage de l'ordinateur
        - Toutefois, il est possible d'écrire sur certains types de mémoire morte, désignés par les sigles PROM (Programmable Read Only Memory), EPROM (Erase Programmable Read Only Memory) et EEPROM (Electrically Erasable Programmable Read Only Memory) :
        - Une PROM est programmée une seule fois, et elle est dédiée à des fonctions spécifiques, comme par exemple des jeux
        - Une EPROM est une EEPROM peuvent être effacées (respectivement par un rayonnement ultraviolet et par un courant électrique) puis reprogrammées pour autre usage

- Les périphériques :
    - Les périphériques sont les organes de l'ordinateur qui permettent de communiquer avec l'unité centrale et de stocker les informations d'une façon permanente
    - Les périphériques d'entrée : le clavier, la souris, le microphone, le scanner, etc …
    - Les périphériques de sortie : l'écran, l'imprimante, le casque, etc …
    - Les périphériques de stockage : le disque dur, le CD-ROM, la clé USB, etc ...

3. La Structure d'un Microprocesseur

- Source : https://www.irif.fr/~carton/Enseignement/Architecture/Cours/Processor

3.1. Composants Principaux

- Tous les micro-processeurs conçus s'organisent globalement de même façon
- Ils peuvent être décomposés en quatre composants décrits par le modèle de von Neumann
- Le modèle de von Neumann donne les quatre composants essentiels qui constituent un micro-processeur
- Il décrit également les interactions entre ces différents composants :

  - l'horloge
  - l'unité de contrôle
  - l'unité de traitement
  - la mémoire
  - l'unité d'entrées/sorties

- l'organisation interne des composants :

  - Le registre d'instruction (IR)
  - le compteur de programme (PC)
  - L'unité de contrôle
  - Les registres mémoires (adresses et données)
  - L'unité arithmétique et logique (ALU)

3.2. Principales Instructions

- Les échanges de données avec la mémoire
- Les opérations arithmétiques : addition, multiplication
- Les opérations de comparaisons : égalité, inférieur, supérieur
- Les opérations logiques : ET, OU, NOT, XOR
- Les opérations de décalages : droite ou gauche sur n bits
- Les opérations répétées en lot : copie, recherche

3.3. Séquencement des Instructions

- Le chargement de l'instruction
- Le décodage de l'instruction
- Le calcul des adresses des opérandes
- Le chargement des opérandes
- L'exécution
- La mise en place du résultat

3.4. Ordonnancement des Instructions

- Le compteur de programme (PC ou Program Counter) :
  - il contient en permanence l'adresse de la prochaine instruction à exécuter
  - à chaque début de cycle d'exécution, l'instruction à exécuter est chargée dans le registre IR à partir de l'adresse contenue dans le registre PC
  - Ensuite, le registre PC est incrémenté pour pointer sur l'instruction suivante
- Le registre d'instruction (IR) :
  - il contient l'instruction en cours d'exécution
  - ce registre est chargé au début du cycle d'exécution par l'instruction dont l'adresse est donnée par le compteur de programme PC

3.5. Codage des Instructions

- Les instructions exécutés par le processeur sont stockées en mémoire
- Toutes les instructions possibles sont représentées par des codes ou op-code
- Les *op-code* peuvent être de longueur fixe ou de longueur variable comme dans le Pentium

4. Introduction au Langage C

4.1. Historique du langage C

- Source Wikipedia
- Le langage C a été inventé au cours de l'année 1972 dans les Laboratoires Bell
- Il était développé en même temps que UNIX par Dennis Ritchie et Ken Thompson
- Ken Thompson avait développé un prédécesseur de C, le langage B, qui est lui-même inspiré de BCPL
- Dennis Ritchie a fait évoluer le langage B dans une nouvelle version suffisamment différente, en ajoutant notamment les types, pour qu'elle soit appelée C1
- Brian Kernighan aida à populariser le langage C
- En 1978, Kernighan fut le principal auteur du livre *The C Programming Language* décrivant le langage enfin stabilisé
- Ritchie s'était occupé des appendices et des exemples avec Unix
- On appelle ce livre « le K&R », et l'on parle de **C traditionnel** ou de **C K&R** lorsqu'on se réfère au langage tel qu'il existait à cette époque
- En 1983, l'Institut national américain de normalisation (ANSI) a formé un comité de normalisation (X3J11) du langage qui a abouti en 1989 à la norme dite **ANSI C** ou **C89** (formellement ANSI X3.159-1989)
- En 1990, cette norme a également été adoptée par l'Organisation internationale de normalisation (**C90**, **C ISO**, formellement ISO/CEI 9899:1990)
- ANSI C est une évolution du C K&R qui reste extrêmement compatible. Elle reprend quelques idées de C++, notamment la notion de prototype et les qualificateurs de type2
- En 1999, une nouvelle évolution du langage est normalisée par l'ISO : Les nouveautés portent notamment sur les tableaux de taille variable, les pointeurs restreints, les nombres complexes, les littéraux composés, les déclarations mélangées avec les instructions, les fonctions *inline*, le support avancé des nombres flottants, et la syntaxe de commentaire de C++
- En 2011, l'ISO ratifie une nouvelle version du standard5 : Cette évolution introduit notamment le support de la programmation multi-thread, les expressions à type générique, et un meilleur support d'Unicode

4.2. Caractéristiques Générales

- Le langage C est un langage de programmation impératif (opérations en séquences d'instructions) et généraliste
- Il est qualifié de langage de bas niveau dans le sens où chaque instruction du langage est conçue pour être compilée en un nombre d'instructions machine assez prévisible en termes d'occupation mémoire et de charge de calcul
- En outre, il propose un éventail de types entiers et flottants conçus pour pouvoir correspondre directement aux types de donnée supportés par le processeur
- Enfin, il fait un usage intensif des calculs d'adresse mémoire avec la notion de pointeur
- le C supporte les types énumérés, composés, et opaques
- Il ne propose en revanche aucune opération qui traite directement des objets de plus haut niveau (fichier informatique, chaîne de caractères, liste, table de hachage…)
- Ces types plus évolués doivent être traités en manipulant des pointeurs et des types composés
- De même, le langage ne propose pas en standard la gestion de la programmation orientée objet, ni de système de gestion d'exceptions
- Il existe des fonctions standards pour gérer les entrées-sorties et les chaînes de caractères, mais contrairement à d'autres langages, aucun opérateur spécifique pour améliorer l'ergonomie
- Ceci rend aisé le remplacement des fonctions standards par des fonctions spécifiquement conçues pour un programme donné
- Ces caractéristiques en font un langage privilégié quand on cherche à maîtriser les ressources matérielles utilisées, le langage machine et les données binaires générées par les compilateurs étant relativement prévisibles
- Ce langage est donc extrêmement utilisé dans des domaines comme la programmation embarquée sur microcontrôleurs, les calculs intensifs, l'écriture de systèmes d'exploitation et les modules où la rapidité de traitement est importante
- Il constitue une bonne alternative au langage d'assemblage dans ces domaines, avec les avantages d'une syntaxe plus expressive et de la portabilité du code source
- Le langage C a été inventé pour écrire le système d'exploitation UNIX, et reste utilisé pour la programmation système
- Ainsi le noyau de grands systèmes d'exploitation comme Windows et Linux sont développés en grande partie en C
- En contrepartie, la mise au point de programmes en C, surtout s'ils utilisent des structures de données complexes, est plus difficile qu'avec des langages de plus haut niveau
- En effet, dans un souci de performance, le langage C impose à l'utilisateur de programmer certains traitements (libération de la mémoire, vérification de la validité des indices sur les tableaux…) qui sont pris en charge automatiquement dans les langages de haut niveau
- C est un langage simple, et son compilateur l'est également. Cela se ressent au niveau du temps de développement d'un compilateur C pour une nouvelle architecture de processeur : Kernighan et Ritchie estimaient qu'il pouvait être développé en deux mois car « on s'apercevra que les 80 % du code d'un nouveau compilateur sont identiques à ceux des codes des autres compilateurs existant déjà

4.3. Qualités et défauts

- C'est un des langages les plus utilisés car :
  - il existe depuis longtemps, le début des années 1970
  - il est fondé sur un standard ouvert
  - de nombreux informaticiens le connaissent
  - il permet la minimisation de l'allocation mémoire nécessaire et la maximisation de la performance, notamment par l'utilisation de pointeurs
  - les compilateurs et bibliothèques logicielles existent sur la plupart des architectures
  - il a influencé de nombreux langages plus récents dont C++, Java, C# et PHP
  - il met en œuvre un nombre restreint de concepts, ce qui facilite sa maîtrise et l'écriture de compilateurs simples et rapides
  - il ne spécifie pas rigidement le comportement du fichier exécutable produit, ce qui aide à tirer parti des capacités propres à chaque ordinateur
  - il permet l'écriture de logiciels qui n'ont besoin d'aucun support à l'exécution (ni bibliothèque logicielle ni machine virtuelle), au comportement prévisible en temps d'exécution comme en consommation de mémoire vive, comme des noyaux de système d'exploitation et des logiciels embarqués

- Ses principaux inconvénients sont :
  - le peu de vérifications offertes par les compilateurs d'origine (K&R C)
  - l'absence de vérifications à l'exécution, ce qui fait que des erreurs qui auraient pu être automatiquement détectées lors du développement ne l'étaient qu'à l'exécution
  - son approche de la modularité restée au niveau de ce qui se faisait au début des années 1970, et largement dépassée depuis par d'autres langages
  - il ne facilite pas la programmation orientée objet
  - la gestion d'exceptions très sommaire
  - le support très limité de la généricité
  - les subtilités de l'écriture de programmes portables, car le comportement exact des exécutables dépend de l'ordinateur cible
  - le support minimaliste de l'allocation de mémoire et des chaînes de caractères, ce qui oblige les programmeurs à s'occuper de détails fastidieux et sources de bugs
  - il n'y a notamment pas de ramasse-miettes standard
  - les bugs graves qui peuvent être causés par un simple manque d'attention du développeur, tel le dépassement de tampon qui constitue une faille de sécurité informatique exploitable par les logiciels malveillants
  - certaines erreurs ne peuvent être détectées automatiquement qu'à l'aide d'outils supplémentaires et non standardisés
  - la faible productivité du langage par rapport aux langages plus récents

4.4. Aperçu de la Syntaxe

- Ce programme *Hello world* est proposé en exemple en 1978 dans *The C Programming Language* de Brian Kernighan et Dennis Ritchie
- Créer un programme affichant *hello world* est depuis devenu l'exemple de référence pour présenter les bases d'un nouveau langage, version de 1978 :

```
main()
{
        printf("hello, world\n");
}
```

  - main est le nom de la fonction principale, aussi appelée point d'entrée du programme
  - les parenthèses () après main indiquent qu'il s'agit d'une fonction
  - les accolades { et } entourent les instructions constituant le corps de la fonction
  - printf est une fonction d'écriture dans la sortie standard
  - les caractères " délimitent une chaîne de caractères, hello, world
  - les caractères \n sont une séquence d'échappement représentant le saut de ligne
  - un point-virgule termine l'instruction expression

- Le même programme suivant les bonnes pratiques contemporaines :

```
#include <stdio.h>

int main(void)
{
   printf("hello, world\n");
   return 0;
}
```

  - #include <stdio.h> inclut l'en-tête standard <stdio.h> contenant les déclarations des fonctions d'entrées-sorties de la bibliothèque standard du C, dont la fonction printf utilisée ici
  - int est le type renvoyé par la fonction main
  - le type int est le type implicite en K&R C et en C89, et il était couramment omis du temps où l'exemple de Kernighan et Ritchie a été écrit, il est obligatoire en C99
  - le mot-clé void entre parenthèses signifie que la fonction n'a aucun paramètre
  - il peut être omis sans ambiguïté lors de la définition d'une fonction
  - l'instruction return 0; indique que la fonction main retourne la valeur 0
  - cette valeur est de type int, et correspond au int devant le main

4.5. Brièveté de la Syntaxe

- La syntaxe de C a été conçue pour être brève
- Elle a souvent été comparée à celle de Pascal, langage impératif également créé dans les années 1970
- Voici un exemple avec une fonction factorielle :

```
/* En C (norme ISO) */
int factorielle(int n) {
    return n > 1 ? n * factorielle(n - 1) : 1;
}
```

```
{ En Pascal }
function factorielle(n: integer) : integer
begin
        if n > 1 then factorielle := n * factorielle(n - 1)
        else factorielle := 1
end
```

- là où Pascal utilise 7 mots clés (function, integer, begin, if, then, else et end), C n'en utilise que 2 (int et return)

4.6. Langage d'Expressions

- La brièveté de C ne repose pas que sur la syntaxe
- Le grand nombre d'opérateurs disponibles, le fait que la plupart des instructions contiennent une expression, que les expressions produisent presque toujours une valeur, et que les instructions de test se contentent de comparer la valeur de l'expression testée avec zéro
- Voici l'exemple de fonction de copie de chaîne de caractères — dont le principe est de copier les caractères jusqu'à avoir copié le caractère nul, qui marque par convention la fin d'une chaîne en C

```
void strcpy(char *s, char *t)
{
        while (*t != '\0') {
                *s = *t;
                s = s + 1; // s ++ ; s += 1;
                t = t + 1;
        }
        *s = *t;
}
```

- Pour comparaison, une version utilisant pas les opérateurs raccourcis et la comparaison implicite à zéro donnerait :

```
void strcpy(char *s, char *t)
{
        while (*s++ = *t++)     ;
}
```

  - la boucle while utilise un style d'écriture classique en C, qui a contribué à lui donner une réputation de langage peu lisible
  - l'expression *s++ = *t++ contient : deux déréférencements de pointeur, deux incrémentations de pointeur, une affectation et la valeur affectée est comparée avec zéro par le while
  - cette boucle n'a pas de corps car toutes les opérations sont effectuées dans l'expression de test du while

5. Des Sources à l'Exécutable

5.1. Sources

- Source Wikipedia
- Un programme écrit en C est généralement réparti en plusieurs fichiers sources compilés séparément
- Les fichiers sources C sont des fichiers texte, généralement dans le codage des caractères du système hôte
- Ils peuvent être écrits avec un simple éditeur de texte
- Il existe de nombreux éditeurs, voire des environnements de développement intégrés (IDE), qui ont des fonctions spécifiques pour supporter l'écriture de sources en C
- L'usage est de donner les extensions de nom de fichier .c et .h aux fichiers source C
- Les fichiers .h sont appelés fichiers d'en-tête, de l'anglais header
- Ils sont conçus pour être inclus au début des fichiers source, et contiennent uniquement des déclarations
- Lorsqu'un fichier .c ou .h utilise un identificateur déclaré dans un autre fichier .h, alors il inclut ce dernier
- Le principe généralement appliqué consiste à écrire un fichier .h pour chaque fichier .c, et à déclarer dans le fichier .h tout ce qui est exporté par le fichier .c

5.2. Génération d'un Exécutable

- La génération d'un exécutable à partir des fichiers sources se fait en plusieurs étapes
- Elles sont souvent automatisées à l'aide d'outils comme make ou bien des outils spécifiques à un environnement de développement intégré
- Les étapes menant des sources au fichier exécutable sont au nombre de quatre : précompilation, compilation, assemblage, édition de liens
- Lorsqu'un projet est compilé, seuls les fichiers .c font partie de la liste des fichiers à compiler
- Les fichiers .h sont inclus par les directives du préprocesseur contenues dans les fichiers source

5.3. Précompilation

- Le préprocesseur C exécute des directives contenues dans les fichiers sources
- Il les reconnaît au fait qu'elles sont en début de ligne, et commencent toutes avec le caractère #
- Parmi les directives les plus courantes, il y a :

  - #include pour l'inclusion
  - #define pour la définition de macro
  - #if pour commencer la compilation conditionnelle
  - #ifdef et #ifndef, équivalents à #if defined et #if ! defined
  - #endif pour clore la compilation conditionnelle

- Outre l'exécution des directives, le préprocesseur remplace les commentaires par un espace blanc, et procède au remplacement des macros
- Pour le reste, le code source est transmis tel quel au compilateur pour la phase suivante
- Il faut toutefois que chaque #include dans le code source soit récursivement remplacé par le code source inclus
- Ainsi, le compilateur reçoit un seul source du préprocesseur, qui constitue l'unité de compilation
- Voici un exemple de fichier source copyarray.h faisant un usage classique des directives du préprocesseur :

```
#ifndef COPYARRAY_H
#define COPYARRAY_H

#include <stddef.h>
        void copyArray(int *, size_t);
#endif
```

  - les directives #ifndef, #define et #endif garantissent que le code à l'intérieur n'est compilé qu'une seule fois même s'il est inclus plusieurs fois
  - la directive #include <stddef.h> inclut l'en-tête qui déclare le type size_t

5.4. Compilation

- La phase de compilation consiste généralement en la génération du code assembleur
- C'est la phase la plus intensive en traitements
- Elle est accomplie par le compilateur proprement dit
- Pour chaque unité de compilation, on obtient un fichier en langage d'assemblage
- Cette étape peut être divisée en sous-étapes :

  - l'analyse lexicale, qui est la reconnaissance des mots clé du langage
  - l'analyse syntaxique, qui analyse la structure du programme et sa conformité avec la norme
  - l'optimisation de code
  - l'écriture d'un code isomorphe à celui de l'assembleur (et parfois du code assembleur lui-même quand cela est demandé en option du compilateur)

- Par abus de langage, on appelle compilation toute la phase de génération d'un fichier exécutable à partir des fichiers sources
- Mais c'est seulement une des étapes menant à la création d'un exécutable

5.5. Assemblage

- Cette étape consiste en la génération d'un fichier objet en langage machine pour chaque fichier de code assembleur
- Les fichiers objet sont généralement d'extension .o sur Unix, et .obj avec les outils de développement pour MS-DOS, Microsoft Windows, VMS, CP/M…
- Cette phase est parfois regroupée avec la précédente par établissement d'un flux de données interne sans passer par des fichiers en langage intermédiaire ou langage d'assemblage
- Dans ce cas, le compilateur génère directement un fichier objet
- Pour les compilateurs qui génèrent du code intermédiaire, cette phase d'assemblage peut aussi être totalement supprimée : c'est une machine virtuelle qui interprétera ou compilera ce langage en code machine natif
- La machine virtuelle peut être un composant du système d'exploitation ou une bibliothèque partagée

5.6. Edition des Liens

- L'édition des liens est la dernière étape, et a pour but de réunir tous les éléments d'un programme
- Les différents fichiers objet sont alors réunis, ainsi que les bibliothèques statiques, pour ne produire qu'un fichier exécutable
- Le but de l'édition de liens est de sélectionner les éléments de code utiles présents dans un ensemble de codes compilés et de bibliothèques, et de résoudre les références mutuelles entre ces différents éléments afin de permettre à ceux-ci de se référencer directement à l'exécution du programme
- L'édition des liens échoue si des éléments de code référencés manquent

6. Eléments du langage

- Source Wikipedia

6.1. Eléments lexicaux

- Le jeu de caractères ASCII suffit pour écrire en C
- Le C est sensible à la casse
- Les caractères blancs (espace, tabulation, fin de ligne) peuvent être librement utilisés pour la mise en page, car ils sont équivalent à un seul espace dans la plupart des cas
- Un programme écrit en C est généralement réparti en plusieurs fichiers sources compilés séparément
- Les fichiers sources C sont des fichiers texte, généralement dans le codage des caractères du système hôte
- Ils peuvent être écrits avec un simple éditeur de texte
- Il existe de nombreux éditeurs, voire des environnements de développement intégrés (IDE)
- L'usage est de donner les extensions de nom de fichier .c et .h aux fichiers source C
- Les fichiers .h sont appelés fichiers d'en-tête, de l'anglais header
- Ils sont conçus pour être inclus au début des fichiers source, et contiennent uniquement des déclarations
- Lorsqu'un fichier .c ou .h utilise un identificateur déclaré dans un autre fichier .h, alors il inclut ce dernier
- Le principe généralement appliqué consiste à écrire un fichier .h pour chaque fichier .c, et à déclarer dans le fichier .h tout ce qui est exporté par le fichier .c

6.2. Mots clés

- Le C89 compte 32 mots clés, dont cinq qui n'existaient pas en K&R C, et qui sont par ordre alphabétique :
- auto, break, case, char, const (C89), continue, default, do, double, else, enum (C89), extern, float, for, goto, if, int, long, register, return, short, signed (C89), sizeof, static, struct, switch, typedef, union, unsigned, void (C89), volatile (C89), while

6.3. Instructions du préprocesseur

- Le préprocesseur du langage C offre les directives suivantes :
- #include, #define, #pragma (C89), #if, #ifdef, #ifndef, #elif (C89), #else, #endif, #undef, #line, #error

6.4. Types

- Le langage C comprend de nombreux types de nombres entiers, occupant plus ou moins de bits
- La taille des types n'est que partiellement standardisée : le standard fixe uniquement une taille minimale et une magnitude minimale
- Les magnitudes minimales sont compatibles avec d'autres représentations binaires que le complément à deux, bien que cette représentation soit presque toujours utilisée en pratique
- Cette souplesse permet au langage d'être efficacement adapté à des processeurs très variés, mais elle complique la portabilité des programmes écrits en C.
- Chaque type entier a une forme « signée » pouvant représenter des nombres négatifs et positifs, et une forme « non signée » ne pouvant représenter que des nombres naturels
- Les formes signées et non signées doivent avoir la même taille.
- Le type le plus commun est int, il représente le mot machine.
- Contrairement à de nombreux autres langages, le type char est un type entier comme un autre, bien qu'il soit généralement utilisé pour représenter les caractères
- Sa taille est par définition d'un octet

6.5. Magnitude des Types

6.5.1. Types entiers

Type	*Taille*	*Magnitude minimale*
`char`	-127 à 127, ou 0 à 255, selon l'implémentation	≥ 8 bits
`signed char`	-127 à 127	
`unsigned char` (C89)	0 à 255	
`short` `signed short`	-32 767 à +32 767	≥ 16 bits
`unsigned short`	0 à 65 535	
`int` `signed int`	-32 767 à +32 767	≥ 16 bits
`unsigned int`	0 à 65 535	
`long` `signed long`	-2 147 483 647 à +2 147 483 647	≥ 32 bits
`unsigned long`	0 à 4 294 967 295	
`long long` (C99) `signed long long` (C99)	-9 223 372 036 854 775 807 à +9 223 372 036 854 775 807	≥ 64 bits
`unsigned long long` (C99)	0 à 18 446 744 073 709 551 615	

- Les types énumérés se définissent avec le mot clé enum

6.5.2. Types à virgule flottante

Type	*Précision*	*Magnitude*
`float`	≥ 6 chiffres décimaux	environ 10^{-37} à 10^{+37}
`double`	≥ 10 chiffres décimaux	environ 10^{-37} à 10^{+37}
`long double`	≥ 10 chiffres décimaux	environ 10^{-37} à 10^{+37}
`long double` (C89)	≥ 10 chiffres décimaux	

- C99 a ajouté float complex, double complex et long double complex, représentant les nombres complexes associés

6.5.3. Types élaborés

- struct, union, * pour les pointeurs
- […] pour les tableaux
- (…) pour les fonctions

6.5.4. Type booléen

- Le type _Bool est standardisé C99, avant il était courant de définir un synonyme :

```
typedef enum boolean { false , true } bool ;
```

6.5.5. Type void

- Le type void représente le vide, comme une liste de paramètres de fonction vide, ou une fonction ne retournant rien
- Le type void* est le pointeur générique : tout pointeur de donnée peut être implicitement converti de et vers void*
- C'est par exemple le type retourné par la fonction standard malloc, qui alloue de la mémoire
- Ce type ne se prête pas aux opérations nécessitant de connaître la taille du type pointé (arithmétique de pointeurs, déréférencement)

6.5.6. Structures

- C supporte les types composés avec la notion de structure
- Pour définir une structure, il faut utiliser le mot-clé **struct** suivi du nom de la structure
- Les membres doivent ensuite être déclarés entre accolades
- Comme toute déclaration, un point-virgule termine le tout
- Les fonctions peuvent recevoir des pointeurs vers des structures
- Ils fonctionnent avec la même syntaxe que les pointeurs classiques
- Néanmoins, l'opérateur `->` doit être utilisé sur le pointeur pour accéder aux champs de la structure
- Il est également possible de déréférencer le pointeur pour ne pas utiliser cet opérateur, et toujours utiliser l'opérateur `.`

6.6. Commentaires

- Dans les versions de C antérieures à C99, les commentaires devaient commencer par une barre oblique et un astérisque (« /* ») et se terminer par un astérisque et une barre oblique
- Presque tous les languages modernes ont repris cette syntaxe pour écrire des commentaires dans le code
- Tout ce qui est compris entre ces symboles est du commentaire, saut de ligne compris :
- La norme C99 a repris de C++ les commentaires de fin de ligne, introduits par deux barres obliques et se terminant avec la ligne :

```
/* Ceci est un commentaire
   sur deux lignes
   ou plus */

// Commentaire jusqu'à la fin de la ligne
```

6.7. Formatage des fonctions printf et annexes

- Une instruction de formatage s'écrit ainsi avec le format précédé de '%', qui sera associé au nom de la variable qu'on veut afficher
- La lettre qui suit les « % » dans le format correspond à un type de variable :

Type	*Lettre*
int	%d / %i
long	%ld
float/double	%f / %lf
char	%c
string (char*)	%s
pointeur (void*)	%p
short	%hd
entier hexadécimal	%x

6.8. Structures de contrôle

- La syntaxe des différentes structures de contrôle existantes en C est largement reprise dans plusieurs autres langages, comme le C++ bien sûr, mais également Java, C#, PHP ou encore JavaScript
- Les trois grands types de structures sont présents :

  - les tests (également appelés branchements conditionnels) avec :

    if (expression) instruction
    if (expression) instruction else instruction

  - switch (expression) instruction, avec case et default dans l'instruction

  - les boucles avec :

    while (expression) instruction
    for (expression_optionnelle ; expression_optionnelle ; expression_optionnelle) instruction
    do instruction while (expression)

  - les sauts (branchements inconditionnels) :

    break
    continue
    return expression_optionnelle
    goto étiquette

6.9. Fonctions

- Les fonctions en C sont des blocs d'instructions, recevant un ou plusieurs arguments et pouvant retourner une valeur
- Si une fonction ne retourne aucune valeur, le mot-clé void est utilisé
- Une fonction peut également ne recevoir aucun argument
- Le mot-clé void est conseillé dans ce cas

```
// Fonction ne retournant aucune valeur (appelée procédure)
void afficher(int a)
{
    printf("%d", a);
}

// Fonction retournant un entier
int somme(int a, int b)
{
    return a + b;
}

// Fonction sans aucun argument
int saisir(void)
{
    int a;
    scanf("%d", &a);
    return a;
}
```

6.10.Prototype

- Un prototype consiste à déclarer une fonction et ses paramètres sans les instructions qui la composent
- Un prototype se termine par un point-virgule

.

```
// Prototype de saisir
int saisir(void);

// Fonction utilisant saisir
int somme(void)
```

- La syntaxe des différentes structures de contrôle existantes en C est largement reprise dans plusieurs autres langages, comme le C++ bien sûr, mais également Java, C#, PHP ou encore JavaScript
- Généralement, tous les prototypes sont écrits dans des fichiers *.h*, et les fonctions sont définies dans un fichier *.c*

```
{
    int a = saisir(), b = saisir();
    return a + b;
}

// Définition de saisir
int saisir(void)
{
    int a;
    scanf("%d", &a);
    return a;
}
```

6.11. Bibliothèques logicielles

6.11.1. La bibliothèque standard

- La bibliothèque standard normalisée, disponible avec toutes les implémentations, présente la simplicité liée à un langage bas-niveau
- Voici une liste de quelques en-têtes déclarant des types et fonctions de la bibliothèque standard :

<assert.h> : pour un diagnostic de conception lors de l'exécution (assert) ;

<ctype.h> : tests et classification des caractères (isalnum, tolower) ;

<errno.h> : gestion minimale des erreurs (déclaration de la variable errno) ;

<math.h> : fonctions mathématiques de base (sqrt, cos) ; nombreux ajouts en C99 ;

<signal.h> : gestion des signaux (signal et raise)

<stddef.h> : définitions générales (déclaration de la constante NULL)

<stdio.h> : pour les entrées/sorties de base (printf, scanf)

<stdlib.h> : fonctions générales (malloc, rand)

<string.h> : manipulation des chaînes de caractères (strcmp, strlen)

<time.h> : manipulation du temps (time, ctime)

- La bibliothèque standard normalisée n'offre aucun support de l'interface graphique, du réseau, des entrées/sorties sur port série ou parallèle, des systèmes temps réel, des processus, ou encore de la gestion avancée des erreurs (comme avec des exceptions structurées)
- Cela pourrait restreindre d'autant la portabilité pratique des programmes qui ont besoin de faire appel à certaines de ces fonctionnalités, sans l'existence de très nombreuses bibliothèques portables
- La gestion de la mémoire n'est pas intégrée au langage mais assurée par des fonctions de la bibliothèque standard

6.11.2. Les bibliothèques externes

- Le langage C étant un des langages les plus utilisés en programmation, de nombreuses bibliothèques ont été créées pour être utilisées avec le C :
- glib, BLAS, etc. Fréquemment, lors de l'invention d'un format de données, une bibliothèque ou un logiciel de référence en C existe pour manipuler le format
- C'est le cas pour zlib, libjpeg, libpng, Expat, les décodeurs de référence MPEG, libsocket

7. Travaux Dirigés

- Voici un exemple de programme travaillé au fur et mesure de la formation :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Déclaration de la structure personne */
struct T_personne
{
        int     naissance;
        int     age;
        char    cp[ 6 ] ;
        char*   nom;
        char*   prenom;
        char*   infos;
};

void anniversaire(struct T_personne * p_personne)
{
   p_personne->age ++;
   printf("Joyeux anniversaire %s pour tes %d ans !\n", (*p_personne).prenom,p_personne-
>age);
}

char* strsplit( char *ch , char ** tb_ch , int * p_nb , char * ch_sep )
{
        char            *pt , *pt_ch , ca ;
        int             nb , nb_max ;

        pt = NULL ;

        if ( ch == NULL || tb_ch == NULL || p_nb == NULL )      goto    return_bad ;

        nb_max = *p_nb ;
        if ( nb_max <= 0 )                      goto    return_bad ;
        nb = 0 ;
        for ( pt = ch , pt_ch = ch ; ca = *pt ; pt ++ )
        {
                if ( strchr( ch_sep , ca ) != NULL )
                {
                        tb_ch[ nb ++ ] = pt_ch ;
                        *pt = 0 ;
                        pt_ch = pt + 1 ;
                }
        }
        tb_ch[ nb ] = pt_ch ;
        if ( nb > nb_max )                      goto    return_bad ;

        *p_nb = nb + 1 ;

        return( NULL ) ;

return_bad :

        return pt ;
}
```

```
int litfichier( char* nom_fichier )
{
	char	ch[ 10000 ] , *pt , *tb_ch[ 10 ] ;
	int	nb_ch ;
	FILE	*ft ;

	ft = NULL ;

	ft = fopen( nom_fichier , "r" ) ;
	if ( ft == NULL)		goto	return_bad ;
	while ( fgets( ch , sizeof( ch ) , ft ) != NULL )
	{
		pt = strpbrk( ch , "\r\n" ) ;
		*pt = 0 ;
		nb_ch = sizeof( tb_ch ) / sizeof( char* ) ;
		pt = strsplit( ch , tb_ch , &nb_ch , ";,|" ) ;
		fprintf( stdout , "nb : %d |%s|%s|%s|\n" , nb_ch , tb_ch[0],tb_ch[1],tb_ch[2] ) ;
	}

	if ( ft != NULL )		fclose( ft ) ;

	return 0 ;

return_bad :

	if ( ft != NULL )		fclose( ft ) ;

	return -1 ;
}

int main()
{
	char	*pt_nom , *pt_prenom , *commentaire , *nom_fichier ;
	struct T_personne personne, *p_personne;

	personne.nom = "DUPONT";
	personne.prenom = "Albert";
	personne.naissance = 1983;
	personne.age = 19;
	personne.infos = NULL ;
	strcpy( personne.cp , "69003" ) ;

	p_personne = &personne ;	//	*(&personne) est équivant à personne
	pt_nom = p_personne->nom ;
	pt_prenom = p_personne->prenom ;

	commentaire = "Ceci est un commentaire ! " ;
//	personne.infos = strdup( commentaire ) ;
	{
		int	lg ;
		lg = strlen( commentaire ) ;
		personne.infos = malloc( lg + 1 ) ;
		memcpy( personne.infos , commentaire , lg + 1 ) ;
//		strcpy( personne.infos , commentaire ) ;
	}

	personne.infos[ 0 ] = 'D' ;

	printf( "nom : %s prenom : %s structure : %s %s\n" , pt_nom ,pt_prenom,
			*(&personne.nom) , *(&personne.prenom) ) ;
```

```
	printf( "nom : %p prenom : %p\n" ,pt_nom ,pt_prenom ) ;
	printf( "structure : %p cp : %p nom : %p prenom : %p\n" , p_personne ,
			&personne.cp , &personne.nom , &personne.prenom ) ;

	printf( "Infos : %s\n" , personne.infos ) ;

	anniversaire( p_personne ) ;
	printf( "Age : %d ans\n",personne.age);
//	anniversaire( &personne ) ;

	nom_fichier = "personnes.txt" ;
	litfichier( nom_fichier ) ;
	printf( "Fichier : %s\n" , nom_fichier ) ;

	if ( 0 )
	{
		int		lg ;
		lg = strlen( commentaire ) ;
		if ( personne.infos != NULL )		free( personne.infos ) ;
		personne.infos = malloc( lg + 1 ) ;
		int			nu , nb ;
		nb = 1000000000 ;
		for ( nu = 0 ; nu < nb ; nu ++ )
		{
			memcpy( personne.infos , commentaire , lg + 1 ) ;
//			strcpy( personne.infos , commentaire ) ;
		}
	}

	if ( personne.infos != NULL )		free( personne.infos ) ;

	return 0 ;
}
```