

A Langage C

La présente annexe liste limitativement les éléments du langage C (norme C99 ou plus récente) dont la connaissance, selon les modalités de chaque sous-section, est exigible des étudiants à la fin de la première année. Ces éléments s'inscrivent dans la perspective de lire et d'écrire des programmes en C; aucun concept sous-jacent n'est exigible au titre de la présente annexe.

À l'écrit, on travaille toujours sous l'hypothèse que les entêtes suivants ont tous été inclus : `<assert.h>`, `<stdbool.h>`, `<stddef.h>`, `<stdint.h>`, `<stdio.h>`, `<stdlib.h>`. Mais ces fichiers ne font pas en soi l'objet d'une étude et aucune connaissance particulière des fonctionnalités qu'ils apportent n'est exigible.

A.1 Traits et éléments techniques à connaître

Les éléments et notations suivants du langage C doivent pouvoir être compris et utilisés par les étudiants sans faire l'objet d'un rappel, y compris lorsqu'ils n'ont pas accès à un ordinateur.

Traits généraux

- Typage statique. Types indiqués par le programme lors de la déclaration ou définition.
- Passage par valeur.
- Délimitation des portées par les accolades. Les retours à la ligne et l'indentation ne sont pas significatifs mais sont nécessaires pour la lisibilité du code.
- Déclaration et définition de fonctions, uniquement dans le cas d'un nombre fixé de paramètres.
- Gestion de la mémoire : pile et tas, allocation statique et dynamique, durée de vie des objets.

Définitions et types de base

- Types entiers signés `int8_t`, `int32_t` et `int64_t`, types entiers non signés `uint8_t`, `uint32_t` et `uint64_t`. Lorsque la spécification d'une taille précise pour le type n'apporte rien à l'exercice, on utilise les types signés `int` et non signé `unsigned int`. Opérations arithmétiques `+`, `-`, `/`, `*`. Opération `%` entre opérandes positifs. Ces opérations sont sujettes à dépassement de capacité. À l'écrit, on élude les difficultés liées à la sémantique des constantes syntaxiques. On ne présente pas les opérateurs d'incrément.
- Le type `char` sert exclusivement à représenter des caractères codés sur un octet. Notation `'\0'` pour le caractère nul.
- Type `double` (on considère qu'il est sur 64 bits). Opérations `+`, `-`, `*`, `/`.
- Type `bool` et les constantes `true` et `false`. Opérateurs `!`, `&&`, `||` (y compris évaluation paresseuse). Les entiers ne doivent pas être utilisés comme booléens, ni l'inverse.
- Opérateurs de comparaison `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Les constantes du programme sont définies par `const type c = v`. On n'utilise pas la directive du préprocesseur `#define` à cette fin.

Types structurés

- Tableaux statiques : déclaration par `type T[s]` où `s` est une constante littérale entière. Lecture et écriture d'un terme de tableau par son indice `T[i]`; le langage ne vérifie pas la licéité des accès. Tableaux statiques multidimensionnels.
- Définition d'un type structuré par `struct nom_s {type1 champ1; ... typen champn};` et ensuite `typedef struct nom_s nom` (la syntaxe doit cependant être rappelée si les étudiants sont amenés à écrire de telles définitions). Lecture et écriture d'un champ d'une valeur de type structure par `v.champ` ainsi que `v->champ`. L'organisation en mémoire des structures n'est pas à connaître.
- Chaînes de caractères vues comme des tableaux de caractères avec sentinelle nulle. Fonctions `strlen`, `strcpy`, `strcat`.

Structures de contrôle

- Conditionnelle `if (c) sT, if (c) sT else sF`.
- Boucle `while (c) s`; boucle `for (init; fin; incr) s`, possibilité de définir une variable dans `init`; `break`.

- Définition et déclaration de fonction, passage des paramètres par valeur, y compris des pointeurs. Cas particuliers : passage de paramètre de type tableau, simulation de valeurs de retour multiples.

Pointeurs et gestion de la mémoire

- Pointeur vers un objet alloué, notation `type* p = &v`. On considère que les pointeurs sont sur 64 bits.
- Déréférencement d'un pointeur valide, notation `*p`. On ne fait pas d'arithmétique des pointeurs.
- Pointeurs comme moyen de réaliser une structure récursive. Pointeur NULL.
- Création d'un objet sur le tas avec `malloc` et `sizeof` (on peut présenter `size_t` pour cet usage mais sa connaissance n'est pas exigible). Libération avec `free`.
- Transtypage de données depuis et vers le type `void*` dans l'optique stricte de l'utilisation de fonctions comme `malloc`.
- En particulier : gestion de tableaux de taille non statiquement connue; linéarisation de tels tableaux quand ils sont multidimensionnels.

Divers

- Utilisation de `assert` lors d'opérations sur les pointeurs, les tableaux, les chaînes.
- Flux standard.
- Utilisation élémentaire de `printf` et de `scanf`. La syntaxe des chaînes de format n'est pas exigible.
- Notion de fichier d'en-tête. Directive `#include "fichier.h"`.
- Commentaires `/* ... */` et commentaires ligne `//`

A.2 Éléments techniques devant être reconnus et utilisables après rappel

Les éléments suivants du langage C doivent pouvoir être utilisés par les étudiants pour écrire des programmes dès lors qu'ils ont fait l'objet d'un rappel et que la documentation correspondante est fournie.

Traits généraux et divers

- Utilisation de `#define`, `#ifndef` et `#endif` lors de l'écriture d'un fichier d'en-tête pour rendre son inclusion idempotente.
- Rôle des arguments de la fonction `int main(int argc, char* argv[])`; utilisation des arguments à partir de la ligne de commande.
- Fonctions de conversion de chaînes de caractères vers un type de base comme `atoi`.
- Définition d'un tableau par un initialiseur $\{t_0, t_1, \dots, t_{N-1}\}$.
- Définition d'une valeur de type structure par un initialiseur $\{.c_1 = v_1, .c_2 = v_2, \dots\}$.
- Compilation séparée.

Gestions des ressources de la machine

- Gestion de fichiers : `fopen` (dans les modes `r` ou `w`), `fclose`, `fscanf`, `fprintf` avec rappel de la syntaxe de formatage.
- Fils d'exécution : inclusion de l'entête `pthread.h`, type `pthread_t`, commandes `pthread_create` avec attributs par défaut, `pthread_join` sans récupération des valeurs de retour.
- Mutex : inclusion de l'entête `pthread.h`, type `pthread_mutex_t`, commandes `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_mutex_destroy`.
- Sémaphore : inclusion de l'entête `semaphore.h`, type `sem_t`, commandes `sem_init`, `sem_destroy`, `sem_wait`, `sem_post`.