Programmer en C

Table des matières

Introduction 2
Variables 2
Affichage 2
Saisie 2
Calculs 2
Fonctions mathématiques 3
Conditions 3
Boucles 4
Boucle while 4
Boucle do… while 4
Boucle for 4
Fonctions 5
Déclaration 5
Exemples et usages 5
Fonctions et variables 5
Composition d'un programme 5
Les pointeurs 6
Intérêts des pointeurs 6
Tableaux 7
Parcourir un tableau 7
Tableau comme paramètre d'une fonction 7
Chaînes de caractères 8
Lettres 8
Chaînes de caractères 8
Manipulation de chaînes de caractères 9
Le préprocesseur 9
Types de variables 10
Enumérations 11
Lecture et écriture dans des fichiers 12
Ouvrir un fichier 12
Ecriture dans un fichier 12
Lecture d'un fichier 12
Positionnement du curseur 13
Opérations supplémentaires 13
Mémoire et allocation dynamique 13

Introduction

Il existe deux types de langages : ceux de haut niveau, permettant d'écrire plus rapidement des programmes et de faire abstraction des caractéristiques du matériel (comme Python ou Ruby) ; et ceux de bas niveau, plus difficiles mais permettant de manipuler le matériel à un degré avancé. Le C fait partie des langages de bas niveau. De plus, c'est un des langages les plus utilisés.

Ce cours a pour but de rassembler succinctement les fonctionnalités, la syntaxe et plus globalement le fonctionnement du langage C. Le cours ainsi que les exemples s'inspirent du cours d'OpenClassrooms sur le C, disponible à l'adresse suivante : https://openclassrooms.com/courses/apprenez-a-programmer-en-c/

Variables

Type	En C
entier	`char`
entier	`int`
entier	`long`
décimal	`float`
décimal	`double`

Constantes (variables non modifiables) :

```
const int NOMBRE_DE_VIES = 5;
```

Variables statiques (accessibles uniquement dans un fichier) :

```
static int resultat = 0;
```

Affichage

Pour afficher un entier : %d. Pour afficher un décimal : %f.

```
printf("Bonjour\n");
printf("Vous avez %d ans", ageUtilisateur);
printf("Vous avez %d ans et %f euros dans votre compte.", ageUtilisateur, montant);
```

`\n` est un caractère spécial et permet un retour à la ligne. De même, `\t` est une tabulation.

Saisie

```
int age = 0;
scanf("%d", &age);
prinf("Vous avez %d ans.", age);
```

Le symbole & avant le nom de la variable est important (voir le chapitre sur les pointeurs). Utiliser `%f` au lieu de `%d` pour saisir un décimal.

Calculs

Opérations basiques (+,-,*,%)
Le type du résultat de la division (/) dépend des types des valeurs : retourne un décimal si les deux nombres sont aussi des décimaux.

Incrémentation :

```
age++; // age = age + 1
age--; // age = age -1
age += 2 // age = age + 2
age *= 8 // age = age * 8
```

Fonctions mathématiques

```
#include <math.h>
```

fabs (pour les décimaux) ; abs (pour les entiers) : valeur absolue

```
double absolu = 0, nombre = -27
absolu = fabs(nombre);
```

Arrondir à l'entier supérieur :

```
ceil(nombre)
```

Arrondir à l'entier inférieur :

```
floor(nombre)
```

Fonction puissance :

```
pow(nombre, puissance)
```

Racine carrée :

```
sqrt(nombre)
```

Conditions

```
if(/* condition */){
    /* code exécuté */
}
else if(/* condition 2 */){
    /* code exécuté */
}
else{
    /* code exécuté */
}
```

Egalité (==) ; différence (!=) ; supérieur (>) ; inférieur (<) ; supérieur ou égal (>=) ; inférieur ou égal (<=); …

```
if(age >= 18){
    printf("Vous êtes majeur");
}
else{
    printf("Vous êtes mineur");
}
```

Possibilité de faire plusieurs conditions à la fois :

ET : &&

OU : ||

NON : !

```
if(age >= 18 && age <= 25){ printf("Vous avez entre 18 et 25 ans"); }
if(!(age >= 18)){ printf("Vous n'avez pas plus de 18 ans."); }
```

Booléens : ce n'est pas un type. 0 vaut faux, et tout nombre différent de 0 est vrai.
Il est alors possible de passer des nombres dans un if :

```
int majeur = 1;
if(majeur){
    printf("Vous êtes majeur");
}
```

switch :

```
switch(age){
    case 2:
        printf("Salut bébé");
        break;
    case 6:
        printf("Salut gamin");
        break;
    default:
        printf("Tu n'as ni 2 ans, ni 6 ans.");
        break;
}
```

Conditions sous forme ternaire :

```
autorisation = (age >= 18)? 1 : 0; // si age est supérieur ou égal à 18, autorisation vaut 1 ; sinon 0.
```

Boucles

Boucle while

```
while(/* condition */){
    /* code exécuté */
}

int nombre = -1;
while(nombre >= 0 || nombre <= 20){
    printf("Entrez un nombre entre 0 et 20");
    scanf("%d", nombre);
}
printf("Vous avez entré le nombre %d", nombre);
```

Boucles infinies :

```
while(1){ boucle infinie */ }
```

Boucle do… while

```
do{
    /* code exécuté au moins une fois */
} while(/* condition */);
```

Boucle for

```
for(initialisation; condition, incrementation){ }
int compteur;
for(compteur = 0; compteur < 10; compteur++){
    printf("Bonjour !\n");
}
```

Fonctions

Déclaration

```
type nomFonction(parametres){
    /* Instructions de la fonction */
}
```

Au début du code (après les `#include`), prototype de la fonction (normalement inclus dans le fichier .h associé, voir chapitre suivant). C'est la première ligne de la fonction, avec un point-virgule à la fin.

```
type nomFonction(parametres);
```

Une fonction peut renvoyer (`type` = `int`, `double`, …) ou non (`type` = `void`) une valeur.

Exemples et usages

```
int triple(int nombre){
    return nombre * 3;
}

int addition(int n1, int n2){
    return n1 + n2;
}

void bonjour(){
    printf("Bonjour");
}
```

Appeler une fonction :

```
int nombre = 23;
nombre = triple(nombre); // triple nombre
bonjour(); // affiche Bonjour
```

Fonctions et variables

Une variable déclarée dans une fonction n'est pas accessible à l'extérieur : c'est la portée des variables. En déclarant une variable statique dans une fonction, la valeur de la variable ne change pas lorsqu'on appelle plusieurs fois la fonction.

Composition d'un programme

Un programme est composé de nombreux fichiers .c (code) et .h (headers, contenant les prototypes). Les fichiers .h sont inclus en haut des fichiers .c à l'aide du préprocesseur. Ces fichiers .c sont transformés en fichiers binaires .o par le compilateur, puis assemblés en un exécutable par le linker.

Les pointeurs

Les pointeurs sont indispensables dans le langage C. Pour faire très court :
A chaque adresse, on peut stocker un seul nombre. Lorsqu'une variable est déclarée et définie, sa valeur est inscrite quelque part en mémoire, dans une adresse précise.

```
int age = 18;
```

age désigne la valeur de la variable (ici, 18)
&age désigne l'adresse de la variable (par exemple, 45003)

Créer une variable de type pointeur :

```
int *monPointeur = NULL;
```

Le pointeur n'a pas de type particulier (comme int, double, …). Le int écrit précédemment correspond au type de la variable dont on souhaite stocker son adresse.

Ici, on réserve une case en mémoire, comme pour une variable. Mais la valeur est faite pour contenir l'adresse d'une autre variable.

```
int *pointeurSurAge = &age;
```

Accéder à la variable associée à l'adresse stockée dans le pointeur :

```
*pointeurSurAge
```

Exemple :

```
printf("%d", *pointeurSurAge); // affichera 18
```

Ne pas confondre l'étoile dans la définition du pointeur et dans l'accès à la variable : le premier permet d'indiquer que l'on souhaite créer un pointeur, le second l'accès à la valeur de la variable.

Intérêts des pointeurs

Un grand intérêt est le passage des pointeurs dans une fonction. Ainsi, on peut passer la référence de variables en paramètres, afin que la fonction puisse directement modifier les variables en mémoire.

Exemple :

```
void triplePointeur(int *pointeurSurNombre);
int main(int argc, char *argv[]){
    int nombre = 5;
    triplePointeur(&nombre); // On envoie l'adresse de nombre à la fonction
    printf("%d", nombre); // On affiche la variable nombre. La fonction a directement
modifié la valeur de la variable car elle connaissait son adresse
    return 0;
}
void triplePointeur(int *pointeurSurNombre){
    *pointeurSurNombre *= 3; // On multiplie par 3 la valeur de nombre
}
```

Les pointeurs permettent aussi de créer des structures de données (tableaux).

Tableaux

Un tableau est une suite de variables d'un même type, situées dans un espace contigu en mémoire (les cases sont les unes à la suite des autres). En définissant un tableau, cet espace mémoire est réservé.

Définition d'un tableau :

```
int tableau[4];
int tableau[4] = {1,34,71,2};
tableau[0] vaut 1
```

Attention : les index commencent à partir de 0, comme en Python (`tableau[0]` est le premier élément, `tableau[1]` le second, …)

```
int tableau[5] = {86,2};
```

La valeur par défaut est 0 (exemple : `tableau[4]` vaut 0)

Attention, ceci est interdit :

```
int taille = 4;
int tableau[taille]; // INTERDIT
```

tableau est un pointeur sur la première case du tableau. Ainsi :

```
printf("%d", tableau); // affiche l'adresse : 2400 par exemple
printf("%d", tableau[1]); // affiche 2
printf("%d", *tableau); // affiche 86
printf("%d", *(tableau+1)); // affiche 2
```

Précisions concernant la dernière ligne : `tableau` est l'adresse de la première case, `tableau + 1` de la seconde case, etc… car les cases sont les unes à la suite des autres.

Parcourir un tableau

On utilise la boucle for :

```
int i = 0;
for(i = 0; i < 5; i++){
    printf("tableau[%d] = %d\n", i, tableau[i]);
}
```

Tableau comme paramètre d'une fonction

Il est nécessaire de connaître la taille du tableau afin de le parcourir. On passe donc en paramètre de la fonction sa taille.

```
void afficherTableau(int tableau[], int tailleTableau){
    int i;
    for(i = 0; i < tailleTableau; i++){
        printf("%d\n", tableau[i]);
    }
}
```

Chaînes de caractères

Lettres

On utilise le type `char` pour stocker une lettre. Ce type ne permet de stocker que des nombres : on utilise donc la table ASCII pour faire la conversion nombres ⇔lettres.

```
char lettre = 'A';
```

Afficher une lettre :

```
printf("%d\n", lettre); // affiche 65
printf("%c\n", lettre); // affiche A
```

On utilise %c pour afficher une lettre.

Saisir une lettre :

```
scanf("%c", &lettre);
```

Chaînes de caractères

Comment stocker une chaîne de caractères ? En utilisant un tableau de type `char`.

```
char chaine[5]; // tableau pouvant contenir 5 caractères
```

Attention : une chaîne de caractères se compose de lettres, et du caractère spécial `\0` à la fin de la chaîne ! Ce caractère permet à l'ordinateur de savoir quand s'arrête la chaîne. Donc, pour stocker la chaîne "Salut", il faut 6 caractères. Cela n'est valable que pour les tableaux de type `char` ; pour les autres tableaux il faudra toujours stocker la taille du tableau.

```
char chaine[6];
chaine[0] = 'S';
chaine[1] = 'a';
chaine[2] = 'l';
chaine[3] = 'u';
chaine[4] = 't';
chaine[5] = '\0';
```

OU

```
char chaine[] = "Salut"; // ne fonctionne que pour l'initialisation
```

Affichage d'une chaîne :

```
printf("%s", chaine); // affiche Salut
```

Saisie d'une chaîne :

```
char prenom[100]; // taille 100, suffisamment grand pour stocker un prénom
scanf("%s", prenom);
printf("Tu t'appelles %s", prenom);
```

Manipulation de chaînes de caractères

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

Calculer la longueur d'une chaîne (sans le \0 final) :

```
strlen(chaine);
```

Copier une chaîne dans une autre :

```
strcpy(copieChaine, chaineACopier);
```

Concaténer deux chaînes :

```
strcat(chaine1, chaine2); // ajoute chaine2 à la suite de chaine1
```

Comparer deux chaînes :

```
strcmp(chaine1, chaine2); // Renvoie 0 si les chaînes sont identiques ; une autre valeur sinon.
```

Rechercher un caractère :

```
strchr(chaine, caractere); // Renvoie un pointeur vers le premier caractère trouvé, NULL sinon.
strrchr(chaine, caractere); // Même chose, mais vers le dernier caractère trouvé.
```

Rechercher une chaîne dans une autre :

```
strstr(chaine, chaineARechercher); // Renvoie un pointeur vers la première occurrence, NULL sinon.
```

Ecrire dans une chaîne :

```
char chaine[100];
int age = 15;
sprintf(chaine, "Tu as %d ans !", age);
// chaine contient "Tu as 15 ans !"
```

Pour toutes les fonctions, vérifiez bien que la chaîne cible est assez grande pour stocker les différentes opérations.

Le préprocesseur

Directives de préprocesseur : exécutées par le préprocesseur avant la compilation.

Inclure une bibliothèque connue :

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

Inclure un fichier :

```
#include "monfichier.h"
```

Une constante de préprocesseur associe une valeur à un mot. Chaque mot trouvé dans le code sera remplacé par la valeur ou le code correspondant.

Définir une constante de préprocesseur :

```
#define TAILLE 300
#define COUCOU() printf("Coucou");
#define MAJEUR(age) if (age >= 18) / printf("Vous êtes majeur\n");
```

Ces constantes sont généralement définies dans les fichiers .h, à côté des prototypes.

Il existe des constantes prédéfinies :

Nom de constante	Valeur
__LINE__	Numéro de la ligne
__FILE__	Nom du fichier actuel
__DATE__	Date de la compilation
__TIME__	Heure de la compilation

```
printf("Ce fichier a été compilé le %s à %s\n", __DATE__, __TIME__);
```

Règle d'usage : pour chaque fichier .h, mettre l'intégralité du fichier dans :

```
#ifndef DEF_NOMDUFICHIER
#define DEF_NOMDUFICHIER
/* Contenu du fichier */
#endif
```

Ce code permet d'inclure le fichier .h qu'une seule fois (et éviter les inclusions de fichiers en boucle). Il fonctionne ainsi :

- Si `DEF_NOMDUFICHIER` n'est pas déclaré, alors on le déclare et le contenu du fichier sera inclus.
- Si on inclut une seconde fois le fichier, `DEF_NOMDUFICHIER` est déjà déclaré, donc le contenu du fichier ne sera pas inclus.

Types de variables

Définir une structure :

```
typedef struct Coordonnees Coordonnees;
struct Coordonnees { // permet de stocker un couple d'entiers, représentant un point
    int x;
    int y;
};
```

```
typedef struct Personne Personne;
struct Personne { // permet de stocker des informations concernant une personne
    char nom[100];
    char prenom[100];
    char adresse[1000];
    int age;
    int sexe;
}
```

Une structure est composée de sous-variables. Elle se définit dans le fichier .h, au même titre que les prototypes de fonctions.

Créer une variable :

```
struct Coordonnees point;
```

La première ligne des codes précédents permet de définir un alias de structure. Ainsi, au lieu d'écrire `struct Coordonnees`, on peut écrire directement `Coordonnees`.

Ainsi, on gagne du temps en écrivant ce code pour créer une variable :

```
Coordonnees point;
```

Accéder à une composante de la structure :

```
point.x = 10;
point.y = 25;

printf("x=%d, y=%d\n", point.x, point.y);
```

Passer une structure à une fonction :

```
int main(int argc, char *argv[]) {
Coordonnees monPoint;
initialiserCoordonnees(&monPoint);
}
```

Fonction :

```
void initialiserCoordonnees(Coordonnees *point) {
    (*point).x = 0;
    (*point).y = 0;
}
```

OU

```
void initialiserCoordonnees(Coordonnees *point) {
    point->x = 0;
    point->y = 0;
}
```

La seconde version est plus rapide et la syntaxe la plus courante. Attention : elle n'est utilisable qu'avec les pointeurs ! En effet, `point->x` revient à écrire `(*point).x`.

Enumérations

```
typedef enum Volume Volume;
enum Volume {
    FAIBLE, MOYEN, FORT
//    0       1     2
};
```

```
typedef enum Volume Volume;
enum Volume {
    MUET = 0, FAIBLE = 10, MOYEN = 50, FORT = 100
};
```

Intérêt des énumérations : rendre le code plus lisible.

```
int volumeMusique;
volumeMusique = MOYEN;
if(volumeMusique == FORT){ }
```

Lecture et écriture dans des fichiers

Ouvrir un fichier

Prototype de la fonction fopen :

```
FILE* fopen(const char *nomFichier, const char *modeOuverture);
```

Mode d'ouverture	Description
r	Lecture seule
w	Ecriture seule
a	Ajout en écriture à la fin du fichier
r+	Lecture et écriture
w+	Lecture et écriture, avec suppression du fichier
a+	Ajout en lecture et écriture à la fin du fichier

Prototype de la fonction fclose :

```
int fclose(FILE* pointeurSurFichier);
```

```
fichier = fopen("fichier.txt", "r+");
if(fichier != NULL){
    /* Opérations sur le fichier */
    fclose(fichier);
}
else{
    printf("Impossible d'ouvrir le fichier");
}
```

Ecriture dans un fichier

Ecrit un seul caractère dans le fichier :

```
fputc('A', fichier);
```

Ecrit une chaîne dans le fichier :

```
fputs("Bonjour à tous !\nJe suis Patrick.", fichier);
```

Ecrit une chaîne dans le fichier :

```
fprintf(fichier, "Patrick a %d ans !", age);
```

Lecture d'un fichier

Lecture caractère par caractère :

```
do {
    caractereActuel = fgetc(fichier);
    printf("%c", caractereActuel);
} while (caractereActuel != EOF);
```

Lecture ligne par ligne :

```
char chaine[TAILLE_MAX] = "";
while (fgets(chaine, TAILLE_MAX, fichier) != NULL){
    printf("%s", chaine);
}
```

Lecture avec fscanf :

```
int score[3] = {0};
fscanf(fichier , "%d %d %d", &score[0], &score[1], & score[2]);
printf("Les meilleurs scores sont : %d, %d et %d", score[0], score[1], score[2]);
```

Positionnement du curseur

Position actuelle du fichier :

```
ftell(fichier);
```

Revenir au début :

```
rewind(fichier);
```

Opérations supplémentaires

Renommer un fichier :

```
rename("ancien.txt", "nouveau.txt");
```

Supprimer définitivement un fichier :

```
remove("fichier.txt");
```

Mémoire et allocation dynamique

Connaître la taille que prend un type de variable (ou structure) :

```
sizeof(int);
```

Pour comprendre le fonctionnement de la mémoire, il faut la découper en cases. La taille d'une variable int est de 4 octets. Elle prendra donc 4 cases en mémoire.
Par conséquent, un tableau de 100 int occupera 4 * 100 = 400 octets en mémoire (même si le tableau est vide : en effet, l'emplacement mémoire est réservé).

Comment demander de la mémoire manuellement ?

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

- Appeler la fonction `malloc` pour demander de la mémoire
- Vérifier la valeur retournée par `malloc`
- Utiliser la mémoire
- Libérer la mémoire avec `free`

Exemple avec une variable de type int :

```
int* memoireAllouee = NULL; // c'est un pointeur
memoireAllouee = malloc(sizeof(int)); // on demande un emplacement de la taille d'un int
if(memoireAllouee == NULL) { exit(0); } // la mémoire n'a pas pu être allouée
printf("Quel âge avez-vous ?");
scanf("%d", memoireAllouee); // on ne met pas & car la variable est déjà un pointeur
printf("Vous avez %d ans\n", *memoireAllouee) ; // on accède à la valeur avec *
free(memoireAllouee); // On libère la mémoire
```

La suite du cours est à venir : allocation dynamique et utilisation avec les tableaux.