CARNUS Enseignement supérieur RODEZ

Langage C

Notions de base

Prof : **Kamal Boudjelaba**

5 octobre 2025

2024–2025

BTS CIEL

Carnus Enseignement Supérieur
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR

Table des matières

1 Introduction **2**
1.1 Bibliothèques de fonctions . . . 2
1.2 Compilation et exécution . . . 2
1.3 Structure minimale . . . 2
2 Types et variables **3**
2.1 Principaux types numériques . . . 3
2.2 Déclaration et initialisation . . . 3
2.3 Constantes . . . 4
3 Opérateurs **5**
4 Lecture et écriture **6**
5 Structures conditionnelles **8**
5.1 `if ... else` . . . 8
5.2 `switch` . . . 8
6 Boucles **9**
6.1 `while` . . . 9
6.2 `do-while` . . . 9
6.3 `for` . . . 9
7 Tableaux **10**
8 Chaînes de caractères **11**
8.1 Fonctions courantes sur les chaînes . . . 12
8.2 Structures . . . 13
9 Fonctions **15**
10 Exercices **18**

1. Introduction

1.1 Bibliothèques de fonctions

En C, les fonctionnalités supplémentaires sont fournies par des **bibliothèques**.
Les fichiers en-tête (`.h`) contiennent les prototypes et définitions nécessaires pour utiliser les fonctions déjà compilées, placées dans des fichiers `.lib` (ou `.a` sous Unix).

— `.c` : code source C
— `.h` : fichier en-tête
— `.o` ou `.obj` : code compilé (objet)
— `.exe` : exécutable
— `.lib` / `.a` : bibliothèque précompilée

Note 1.1 :

Pour utiliser une fonction, le compilateur doit la connaître (#include du bon fichier .h) et l'éditeur de liens doit savoir où trouver son code machine (bibliothèque associée).

Conventions de codage

Dans ce cours, nous adoptons les conventions suivantes :
— Indentation : 4 espaces (ou la touche Tab)
— Accolades : style K&R (accolade ouvrante sur la même ligne)
— Noms de variables : `valeur_moy` en minuscules
— Noms de constantes : `VALEUR_MAX_MIN`
— Noms de fonctions : `calculer_moyenne` avec verbe d'action

1.2 Compilation et exécution

```
# Compilation
gcc nom_fichier.c -o nom_programme

# Exécution
./nom_programme          # Linux/Mac
nom_programme.exe        # Windows
```

1.3 Structure minimale

```c
#include <stdio.h>

int main(void) {
    printf("Bonjour, monde !\n");
    return 0;
}
```

Séquences d'échappement utiles :

```
\t : tabulation (horizontale)
\v : tabulation (verticale)
\n : nouvelle ligne
\b : backspace (efface le caractère précédent)
\r : retour (retour au début de ligne, sans saut de ligne : retour chariot)
```

```
\f : saut de page
\a : bip sonore
\\ : antislash
\" : guillemet double
```

Exercice 1.1: Affichage simple

Modifier le programme minimal pour qu'il affiche trois lignes : votre nom, votre prénom et votre ville de naissance.

2. Types et variables

2.1 Principaux types numériques

Type	Signification	Remarques
`char`	Caractère / petit entier	Signé ou non signé selon implémentation
`int`	Entier standard	Taille dépend architecture (16/32/64 bits)
`long`	Entier long	4 octets
`float`	Réel simple précision	6-7 chiffres
`double`	Réel double précision	15 chiffres

- `signed char c1; // -128 à 127`
- `unsigned char c2; // 0 à 255`

Note 2.1 :

Toujours vérifier la taille réelle avec `sizeof(type)` : `printf("sizeof(int) = %zu", sizeof(int));`

2.2 Déclaration et initialisation

```c
int age = 25;
float x, y;
double p = 2.5743926535;
```

Exercice 2.1: Librairie math.h

Écrire un programme C qui utilise la fonction sqrt de math.h pour calculer la racine carrée d'un nombre saisi par l'utilisateur.

Note 2.2 :

Sous Linux/Unix, il faut compiler avec l'option `-lm` : `gcc prog.c -o prog -lm`

Exercice 2.2: Type de variables

Déclarer et initialiser une variable de chaque type vu dans le tableau. Afficher ensuite leur valeur et leur taille (en octets) avec sizeof.

Corrigé

```c
#include <stdio.h>

int main(void) {
    char c = 'A';
    int i = 42;
    long l = 123456L;
    float f = 3.14f;
    double d = 2.718281828;

    printf("char c = %c, taille = %zu octets\n", c, sizeof(c));
    printf("int i = %d, taille = %zu octets\n", i, sizeof(i));
    printf("long l = %ld, taille = %zu octets\n", l, sizeof(l));
    printf("float f = %f, taille = %zu octets\n", f, sizeof(f));
    printf("double d = %lf, taille = %zu octets\n", d, sizeof(d));

    return 0;
}
```

2.3 Constantes

En C, une constante est une donnée dont la valeur ne peut pas être modifiée pendant l'exécution du programme. On peut déclarer une constante avec le mot-clé const ou à l'aide de la directive de préprocesseur define.

```c
// Méthode 1 : const
const double PI = 3.14159;  // Point-virgule nécessaire
const int taille = 100;  // Point-virgule nécessaire

// Méthode 2 : #define
#define TAILLE 100        // Pas de point-virgule !
```

Note 2.3 :

La directive define est traitée avant la compilation par le préprocesseur, tandis que const est gérée par le compilateur et permet de mieux typer la valeur.

Note 2.4 : Convention de nommage

Par convention, les constantes #define s'écrivent en MAJUSCULES, tandis que les constantes const suivent les conventions des variables.

Exercice 2.3: Les constantes

Créer un programme qui définit des constantes pour le nombre de jours dans une semaine et le nombre d'heures dans une journée, puis calcule et affiche le nombre total d'heures dans une semaine.

3. Opérateurs

Le langage C fournit différents opérateurs : arithmétiques, de comparaison, logiques, d'affectation, etc.

Symbole	Signification	Exemple
=	Affectation	a = 5 ;
+	Addition	a + b
-	Soustraction	a - b
*	Multiplication	a * b
/	Division entière ou flottante	a / b
%	Modulo (reste de la division entière)	a % b
==	Égalité	a == b
!=	Différent	a != b
<, <=	Inférieur, Inférieur ou égal	a < b
>, >=	Supérieur, Supérieur ou égal	a >= b
&&	ET logique	(x>0 && y<0)
||	OU logique	(a==0 || b==0)
!	NON logique	!a
++	Incrément	i++
–	Décrément	–j

```c
int i = 5;
printf("%d\n", i++);  // Affiche 5, puis i devient 6
printf("%d\n", ++i);  // i devient 7, puis affiche 7
```

Note 3.1 : Division – précision importante

La division entre deux entiers donne un résultat entier (division euclidienne) :

```c
int a = 7, b = 2;
printf("%d\n", a / b);  // Affiche 3, pas 3.5 !
printf("%f\n", (double)a / b);  // Affiche 3.5 (cast)
```

4. Ordre de priorité

Note 3.2 :

Ordre de priorité (du plus fort au plus faible) :

- Parenthèses : `()`
- Unaires : `!`, `++`, `-`
- Multiplicatifs : `*`, `/`, `%`
- Additifs : `+`, `-`
- Comparaison : `<`, `<=`, `>`, `>=`
- Égalité : `==`, `!=`
- Logiques : `&&`, `||`

Exercice 3.1: Opérateurs arithmétiques

Demander deux entiers à l'utilisateur, puis afficher la somme, la différence, le produit, le quotient entier et le reste.
Attention : vérifier que le diviseur n'est pas nul avant la division.

4. Lecture et écriture

Formats valides pour `printf` et `scanf`

```
%d ou %i : entier signé
%u : entier non signé
%f : flottant (float) et double pour printf
%lf : double pour scanf uniquement
%c : caractère
%s : chaîne
```

Exemple 4.1

```c
#include <stdio.h>

int main(void) {
    int age;
    printf("Quel âge avez-vous ? ");
    scanf("%d", &age);
    printf("Vous avez %d ans.\n", age);
    return 0;
}
```

Note 4.1 :

`gets()` est obsolète et dangereuse (risque de débordement mémoire). Utiliser `fgets()`.

Exemple 4.2

```c
#include <stdio.h>

int main(void) {
    char nom[20];
    printf("Votre nom ? ");
    fgets(nom, sizeof(nom), stdin);
    printf("Bonjour %s", nom);
    return 0;
}
```

Spécificateur	Signification	Type C	Exemple
%d / %i	Entier signé	int	-42, 123
%u	Entier non signé	unsigned int	42
%ld / %li	Long signé	long int	-123456L
%lu	Long non signé	unsigned long	123456UL
%f	Réel simple ou double (printf)	float/double	3.14
%lf	Double (scanf uniquement)	double	2.7182818
%Lf	Long double	long double	1.2345L
%c	Caractère	char	'A'
%s	Chaîne de caractères	char*	"Bonjour"
%p	Adresse mémoire	void*	0x7ffd23c2
%o	Octal	int	077
%x / %X	Hexadécimal	int	0xff / 0xFF

Note 4.2 : scanf

`scanf` laisse le caractère `\n` dans le buffer. Si vous utilisez `fgets` après `scanf`, pensez à vider le buffer :

```c
int age;
char nom[20];
scanf("%d", &age);
while(getchar() != '\n');  // Vide le buffer
fgets(nom, sizeof(nom), stdin);
```

Note 4.3 : fgets

`fgets()` conserve le caractère '\n' en fin de chaîne. Pour le supprimer :

```c
char nom[20];
fgets(nom, sizeof(nom), stdin);
nom[strcspn(nom, "\n")] = '\0';  // Supprime le \n
```

5. Structures conditionnelles

Note 5.1 :

Les accolades `{...}` sont optionnelles si le bloc ne contient qu'une instruction, mais recommandées pour éviter les erreurs :

```c
// Correct mais déconseillé
if (x > 0)
    printf("Positif\n");

// Recommandé
if (x > 0) {
    printf("Positif\n");
}
```

5.1 if ... else

```c
if (score >= 10)
    printf("Réussi\n");
else
    printf("Échoué\n");
```

Exercice 5.1: Conditions

Écrire un programme qui lit un entier et affiche s'il est pair ou impair.

Corrigé

```c
#include <stdio.h>

int main(void) {
    int n;
    printf("Entrez un entier : ");
    scanf("%d", &n);

    if (n % 2 == 0) {
        printf("%d est pair\n", n);
    } else {
        printf("%d est impair\n", n);
    }

    return 0;
}
```

5.2 switch

```c
switch(jour) {
    case 1: printf("Lundi"); break;
    case 2: printf("Mardi"); break;
    default: printf("Jour invalide");
}
```

Note

Pensez à mettre `break;` dans chaque `case` pour éviter l'exécution des blocs suivants de manière involontaire (fall-through).

6. Boucles

Les boucles permettent de répéter un bloc d'instructions tant qu'une condition est vérifiée.

6.1 while

```c
int n = 10;
while (n > 0) {
    printf("%d\n", n);
    n--;
}
```

6.2 do-while

```c
int c = 0;
do {
    printf("Valeur de c : %d\n", c);
    c++;
} while (c < 3);
```

Note 6.1 : Différence while vs do-while

Différence importante :

- `while` : condition testée **avant** l'exécution (peut ne jamais s'exécuter)
- `do-while` : condition testée **après** (s'exécute au moins une fois)

Attention

Vérifiez toujours que votre condition finira par devenir fausse, sinon vous créez une **boucle infinie** :

```c
int i = 0;
while (i < 10) {
    printf("%d\n", i);
    // Oubli du i++      boucle infinie !
}
```

6.3 for

```c
for (int i = 0; i < 5; i++)
    printf("%d ", i);
```

Instructions `break` et `continue`

- `break` : sort immédiatement de la boucle
- `continue` : passe directement à l'itération suivante

```c
for (int i = 0; i < 10; i++) {
    if (i == 5) continue;  // Saute l'affichage de 5
    if (i == 8) break;     // Arrête à 8
    printf("%d ", i);      // Affiche : 0 1 2 3 4 6 7
}
```

Exercice 6.1: Boucle for

Afficher les 10 premiers multiples de 7 à l'aide d'une boucle for.

7. Tableaux

Un tableau est une structure qui stocke un ensemble de valeurs de même type, contiguës en mémoire.

Initialisation partielle et complète

```c
int t[5] = {1, 2}; // Les autres sont initialisés à 0
int tab[5] = {1, 2, 3, 4, 5};
```

DANGER

Le C ne vérifie PAS les dépassements de tableau !

```c
int t[5] = {1, 2, 3, 4, 5};
printf("%d\n", t[10]);  // ERREUR : comportement indéfini !
```

C'est une source majeure de bugs. Soyez toujours vigilants sur les indices.

Note 7.1 :

Pour obtenir le nombre d'éléments d'un tableau :

```c
int t[5] = {1, 2, 3, 4, 5};
int taille = sizeof(t) / sizeof(t[0]);  // taille = 5
```

Attention : ceci ne fonctionne que dans la fonction où le tableau est déclaré !

Parcours de tableau

```c
#include <stdio.h>

int main(void) {
    int tab[5] = {10, 20, 30, 40, 50};

    // Parcours avec for
    for (int i = 0; i < 5; i++) {
        printf("tab[%d] = %d\n", i, tab[i]);
    }

    return 0;
}
```

Tableaux 2D :

```c
int m[2][3] = {{1,2,3}, {4,5,6}};

// Accès aux éléments
printf("%d\n", m[0][1]);  // Affiche 2 (ligne 0, colonne 1)

// Parcours
for (int i = 0; i < 2; i++) {       // lignes
    for (int j = 0; j < 3; j++) {  // colonnes
        printf("%d ", m[i][j]);
    }
    printf("\n");
}
```

Note 7.2 :

En mémoire, les éléments sont stockés ligne par ligne (row-major order) : `m[0][0]`, `m[0][1]`, `m[0][2]`, `m[1][0]`, `m[1][1]`, `m[1][2]`

Exercice 7.1: Tableaux

Demander à l'utilisateur de saisir 5 entiers dans un tableau, puis afficher la somme de ces entiers.

Corrigé

Le code devrait ressembler à :

```c
#include <stdio.h>

int main(void) {
    int tab[5];
    int somme = 0;

    printf("Saisissez 5 entiers :\n");
    for (int i = 0; i < 5; i++) {
        printf("Entier %d : ", i+1);
        scanf("%d", &tab[i]);
        somme += tab[i];
    }

    printf("Somme = %d\n", somme);
    return 0;
}
```

8. Chaînes de caractères

En C, une chaîne de caractères est un tableau de char se terminant par le caractère spécial '\0' (zéro terminal).

```c
char prenom[20];
fgets(prenom, sizeof(prenom), stdin);

char nom[] = "Carnus";
printf("Bonjour %s\n", nom);
```

Note 8.1 : Différence entre déclaration et initialisation

Différence importante :

```c
// Initialisation à la déclaration : OK
char nom[] = "Carnus";  // Taille calculée automatiquement (7 octets)

// Affectation impossible :
char nom[20];
nom = "Carnus";  // ERREUR : on ne peut pas affecter un tableau !

// Solution : utiliser strcpy
char nom[20];
strcpy(nom, "Carnus");  // Nécessite #include <string.h>
```

Note 8.2 :

Les chaînes de caractères sont terminées par le caractère spécial '\0' ("zéro" binaire, non imprimable).
Le zéro terminal permet aux fonctions de manipulation de chaînes (comme strlen, strcpy, etc.) de savoir où la chaîne se termine.

8.1 Fonctions courantes sur les chaînes

Fonction	Description
`strlen(s)`	Retourne la longueur de la chaîne (sans compter `\0`)
`strcpy(dest, src)`	Copie `src` dans `dest`
`strncpy(dest, src, n)`	Copie au plus `n` caractères (plus sûr)
`strcat(dest, src)`	Concatène `src` à la fin de `dest`
`strcmp(s1, s2)`	Compare deux chaînes (retourne 0 si égales)
`strchr(s, c)`	Cherche le caractère `c` dans `s`
`strstr(s1, s2)`	Cherche la sous-chaîne `s2` dans `s1`

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

int main(void) {
    char nom[20] = "Charles";

    printf("Longueur : %zu\n", strlen(nom));  // 5

    if (strcmp(nom, "Charles") == 0) {
        printf("Les chaînes sont identiques\n");
    }

    return 0;
}
```

Note 8.3 :

`fgets` conserve le caractère de nouvelle ligne `\n` en fin de chaîne. Pour le supprimer :

```c
char prenom[20];
fgets(prenom, sizeof(prenom), stdin);

// Méthode 1 : avec strcspn
prenom[strcspn(prenom, "\n")] = '\0';

// Méthode 2 : avec strlen
size_t len = strlen(prenom);
if (len > 0 && prenom[len-1] == '\n') {
    prenom[len-1] = '\0';
}
```

Exercice 8.1: Chaînes de caractères

Demander à l'utilisateur de saisir son prénom, puis affichez "Bonjour, prénom !".

Corrigé

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

int main(void) {
    char prenom[50];

    printf("Entrez votre prénom : ");
    fgets(prenom, sizeof(prenom), stdin);

    // Suppression du \n
    prenom[strcspn(prenom, "\n")] = '\0';
```

```
    printf("Bonjour, %s !\n", prenom);

    return 0;
}
```

8.2 Structures

Les structures permettent de regrouper plusieurs variables de types éventuellement différents sous un même nom.

```
struct Point {
    int x;
    int y;
};

struct Point p1 = {10, 20};
```

Structures avec typedef

```
// Sans typedef : répétitif
struct Point {
    int x;
    int y;
};
struct Point p1;

// Avec typedef : plus concis
typedef struct {
    int x;
    int y;
} Point;

Point p1;  // Plus simple !
```

Accès aux membres

```
typedef struct {
    int x;
    int y;
} Point;

// Avec une variable
Point p1 = {10, 20};
printf("x = %d, y = %d\n", p1.x, p1.y);  // Opérateur .
```

Tableaux de structures

```
typedef struct {
    char nom[50];
    int age;
} Personne;

Personne classe[3] = {
    {"Alice", 20},
    {"Bob", 22},
    {"Charlie", 21}
};

for (int i = 0; i < 3; i++) {
    printf("%s a %d ans\n", classe[i].nom, classe[i].age);
}
```

Structures imbriquées

```
typedef struct {
    int jour;
    int mois;
    int annee;
```

```
} Date;

typedef struct {
    char nom[50];
    Date naissance;
} Personne;

Personne p = {"Alice", {15, 3, 2000}};
printf("Née le %d/%d/%d\n",
       p.naissance.jour,
       p.naissance.mois,
       p.naissance.annee);
```

Exercice 8.2:

Définir une structure Livre contenant un titre, un auteur et un prix. Demander à l'utilisateur de saisir les informations d'un livre et afficher-les.

Corrigé

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

typedef struct {
    char titre[100];
    char auteur[50];
    float prix;
} Livre;

int main(void) {
    Livre livre;

    printf("Titre du livre : ");
    fgets(livre.titre, sizeof(livre.titre), stdin);
    livre.titre[strcspn(livre.titre, "\n")] = '\0';

    printf("Auteur : ");
    fgets(livre.auteur, sizeof(livre.auteur), stdin);
    livre.auteur[strcspn(livre.auteur, "\n")] = '\0';

    printf("Prix : ");
    scanf("%f", &livre.prix);

    printf("\n=== Informations du livre ===\n");
    printf("Titre : %s\n", livre.titre);
    printf("Auteur : %s\n", livre.auteur);
    printf("Prix : %.2f €\n", livre.prix);

    return 0;
}
```

9. Fonctions

Une fonction permet d'encapsuler un morceau de code réutilisable.
Elle a une signature (type de retour, nom, paramètres) et un corps (instructions).

Différence **déclaration** / **définition** :

```c
int somme(int a, int b); // prototype (Déclaration)

int somme(int a, int b) { // définition
    return a + b;
}
```

Portée des variables

```c
int global = 10;  // Variable globale (éviter si possible)

void fonction() {
    int local = 5;  // Variable locale
    printf("%d\n", global);  // OK : accès à global
    printf("%d\n", local);   // OK : accès à local
}

int main(void) {
    printf("%d\n", global);  // OK : accès à global
    // printf("%d\n", local);  // ERREUR : local n'existe pas ici
    return 0;
}
```

Note 9.1 :

Privilégiez les variables locales. Les variables globales rendent le code difficile à maintenir.

Exercice 9.1: Les fonctions

Écrire une fonction "carre" qui prend un entier en paramètre et retourne son carré. Tester-la dans le main.

Corrigé

```c
#include <stdio.h>

int carre(int n) {
    return n * n;
}

int main(void) {
    int x;
    printf("Entrez un entier : ");
    scanf("%d", &x);
    printf("Le carré de %d est %d\n", x, carre(x));
    return 0;
}
```

Fonctions récursives

Exemple 9.1

```c
#include <stdio.h>

long facto(int n) {
    if (n <= 1) return 1;
    return n * facto(n - 1);
}

int main(void) {
    int num = 4;
    printf("Factorielle de %d = %ld\n", num, facto(num));
    return 0;
}
```

Appel récursif de facto(4):

```
+--------------------+   <- Haut de pile
| facto(1)           |   Cas de base: retourne 1
+--------------------+
| facto(2)           |   Attend retour de facto(1)
+--------------------+
| facto(3)           |   Attend retour de facto(2)
+--------------------+
| facto(4)           |   Attend retour de facto(3)
+--------------------+
```

Résumé

Structure minimale

```c
#include <stdio.h>

int main(void) {
    printf("Bonjour\n");
    return 0;
}
```

Types principaux

Type	Taille typique	Description
char	1 octet	caractère / petit entier (-128 à 127)
int	4 octets	entier standard (-210 à 210)
long	4-8 octets	entier long
float	4 octets	réel simple précision (6-7 chiffres)
double	8 octets	réel double précision (15 chiffres)

Opérateurs

Affectation	=
Arithm.	+, -, *, /, % (reste)
Compar.	==, !=, <, <=, >, >=
Logiques	&&, ||, !
Incr./Décr.	++, –

Formats printf / scanf

%d / %i	entier signé	int
%u	entier non signé	unsigned int
%f	flottant	float
%lf	double	double
%c	caractère	char
%s	chaîne	char*
%p	adresse	void*

Lecture / Écriture

```
printf("x = %d", x);
scanf("%d", &x);
fgets(chaine, sizeof(chaine), stdin);
```

Structures conditionnelles

```
if (cond) { ... } else { ... }

switch (var) {
    case 1: ...; break;
    default: ...;
}
```

Boucles

```
while (cond) { ... }

do { ... } while (cond);

for (int i=0; i<n; i++) { ... }
```

Tableaux

```
int t[5] = {1, 2, 3, 4, 5};
printf("%d", t[0]);

int m[2][3] = {{1,2,3},{4,5,6}};
```

Chaînes de caractères

```
char nom[20] = "Alice";
fgets(nom, sizeof(nom), stdin);
```

Structures

```
typedef struct {
    char nom[50];
    int age;
} Personne;

Personne p = {"Alice", 20};
printf("%s a %d ans\n", p.nom, p.age);
```

Fonctions

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

10. Exercices

Exercice 1

Ecrire un programme C qui lit trois entiers et affiche leur somme, leur produit et leur moyenne.
Correction :

```c
#include <stdio.h>

int main(void) {
    int a, b, c;

    printf("Entrez trois entiers : ");
    scanf("%d %d %d", &a, &b, &c);

    int somme = a + b + c;
    int produit = a * b * c;
    double moyenne = (double)(a + b + c) / 3;

    printf("Somme : %d\n", somme);
    printf("Produit : %d\n", produit);
    printf("Moyenne : %.2f\n", moyenne);

    return 0;
}
```

Note 10.1 :

Le cast `(double)` est nécessaire pour obtenir une division flottante et non entière.

Exercice 2

Ecrire un programme C qui lit trois entiers A, B, et C et affiche le maximum et le minimum.
Correction :

```c
#include <stdio.h>

int main(void) {
    int a, b, c;

    printf("Entrez trois entiers : ");
    scanf("%d %d %d", &a, &b, &c);

    // Recherche du maximum
    int max = a;
    if (b > max) max = b;
    if (c > max) max = c;

    // Recherche du minimum
    int min = a;
    if (b < min) min = b;
    if (c < min) min = c;

    printf("Maximum : %d\n", max);
    printf("Minimum : %d\n", min);

    return 0;
}
```

Alternative avec opérateur ternaire :

```c
int max = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);
int min = (a < b) ? ((a < c) ? a : c) : ((b < c) ? b : c);
```

Exercice 3

Soit la suite U_n définie par : $U_0 = 1$ et $U_{n+1} = 5U_n + 3$.
Ecrire un programme qui permet de lire n et de calculer U_n.

Correction :

```c
#include <stdio.h>

int main(void) {
    int n;
    printf("Entrez le rang n : ");
    scanf("%d", &n);

    if (n < 0) {
        printf("n doit être positif ou nul\n");
        return 1;
    }

    long long u = 1;  // U_0 = 1

    for (int i = 0; i < n; i++) {
        u = 5 * u + 3;  // U_{n+1} = 5*U_n + 3
    }

    printf("U_%d = %lld\n", n, u);

    return 0;
}
```

Note 10.2 :

On utilise `long long` car la suite croît très rapidement. Par exemple : $U_{10} = 30517578$ et $U_{15} > 10^{10}$.

Version récursive (bonus) :

```c
long long suite_u(int n) {
    if (n == 0) return 1;
    return 5 * suite_u(n - 1) + 3;
}
```

Exercice 4

En utilisant `switch`, écrire un programme qui :

- demande à l'utilisateur d'entrer deux nombres réels,
- demande de choisir l'opération à réaliser ('+' pour addition, '-' pour soustraction, '*' pour multiplication, '/' pour division),
- affiche le résultat de l'opération,
- gère le cas de la division par zéro.

Correction :

```c
#include <stdio.h>

int main(void) {
    double a, b, resultat;
    char operation;

    printf("Entrez deux nombres : ");
    scanf("%lf %lf", &a, &b);

    printf("Choisissez l'opération (+, -, *, /) : ");
    scanf(" %c", &operation);  // Espace avant %c pour ignorer \n

    switch(operation) {
        case '+':
```

```c
            resultat = a + b;
            printf("%.2f + %.2f = %.2f\n", a, b, resultat);
            break;

        case '-':
            resultat = a - b;
            printf("%.2f - %.2f = %.2f\n", a, b, resultat);
            break;

        case '*':
            resultat = a * b;
            printf("%.2f * %.2f = %.2f\n", a, b, resultat);
            break;

        case '/':
            if (b == 0) {
                printf("Erreur : division par zéro !\n");
            } else {
                resultat = a / b;
                printf("%.2f / %.2f = %.2f\n", a, b, resultat);
            }
            break;

        default:
            printf("Opération invalide !\n");
    }

    return 0;
}
```

Note 10.3 :

L'espace avant `%c` dans `scanf(" %c", &operation)` permet d'ignorer les espaces et le `\n` laissé par le `scanf` précédent.

Exercice 5

Analyser les trois codes suivants qui tentent de gérer une division par zéro.
Pour chacun, identifier s'il est correct, et expliquer son comportement.

Code 1 :

```c
#include <stdio.h>

int main(void) {
    int x = 1;
    float a;

    while (x <= 10) {
        if (x == 7) {
            printf("Division par zéro !");
            continue;  // Problème ici
        }
        a = 1.0 / (x - 7);
        printf("%f\n", a);
        x++;
    }
    return 0;
}
```

Analyse :
Code incorrect : Boucle infinie !
Quand `x == 7`, `continue` fait repartir la boucle sans exécuter `x++`. La variable `x` reste donc bloquée à 7 indéfiniment.
Correction :

```c
if (x == 7) {
    printf("Division par zéro !\n");
    x++;  //     Incrémenter AVANT continue
    continue;
}
```

Code 2 :

```c
#include <stdio.h>

int main(void) {
    int x = 1;
    float a;

    while (x <= 10) {
        if (x == 7) {
            printf("Division par 0\n");
            x++;
            continue;
        }
        a = 1.0 / (x - 7);
        printf("%f\n", a);
        x++;
    }
    return 0;
}
```

Analyse :
Code correct : Pas de boucle infinie.
L'incrémentation `x++` est faite avant `continue`, donc la boucle progresse normalement. Le programme affiche un message pour `x == 7` puis continue avec `x == 8`.
Sortie :

```
-0.166667  (1/(1-7))
-0.200000  (1/(2-7))
-0.250000  (1/(3-7))
-0.333333  (1/(4-7))
-0.500000  (1/(5-7))
-1.000000  (1/(6-7))
Division par 0
0.333333   (1/(8-7))
0.250000   (1/(9-7))
0.200000   (1/(10-7))
```

Code 3 :

```c
#include <stdio.h>

int main(void) {
    int x;
    float a;

    for (x = 1; x <= 10; x++) {
        a = x - 7;
        if (a == 0) {
            printf("Division par 0\n");
            break;
        }
        printf("%f\n", 1.0 / a);
    }
    return 0;
}
```

Analyse :
Code correct mais comportement différent.
Utilise `break` qui **arrête complètement** la boucle dès que `x == 7`. Contrairement au Code 2, les valeurs pour `x = 8, 9, 10` ne sont pas calculées.
Sortie :

```
-0.166667
-0.200000
-0.250000
-0.333333
-0.500000
-1.000000
Division par 0
(arrêt de la boucle)
```

Conclusion

- `continue` : saute le reste de l'itération actuelle et passe à la suivante
- `break` : sort complètement de la boucle
- Toujours s'assurer que les variables de contrôle sont bien mises à jour avant `continue`

Exercice 6 : Mini-projets

6.1 - Compteur de mots

Lire une ligne avec `fgets()` et compter le nombre de mots (un mot est une séquence de caractères séparés par des espaces).
Correction :

```c
#include <stdio.h>
#include <ctype.h>

int main(void) {
    char ligne[256];
    int nb_mots = 0;
    int dans_mot = 0;

    printf("Entrez une ligne : ");
    fgets(ligne, sizeof(ligne), stdin);

    for (int i = 0; ligne[i] != '\0'; i++) {
        if (isspace(ligne[i])) {
            dans_mot = 0;
        } else if (dans_mot == 0) {
            dans_mot = 1;
            nb_mots++;
        }
    }

    printf("Nombre de mots : %d\n", nb_mots);
    return 0;
}
```

6.2 - Inversion d'un tableau

Implémenter une fonction qui inverse un tableau d'entiers.
Correction :

```c
#include <stdio.h>

void inverser(int tab[], int taille) {
    int debut = 0;
    int fin = taille - 1;
```

```
    while (debut < fin) {
        // Échange des éléments
        int temp = tab[debut];
        tab[debut] = tab[fin];
        tab[fin] = temp;

        debut++;
        fin--;
    }
}

void afficher(int tab[], int taille) {
    for (int i = 0; i < taille; i++) {
        printf("%d ", tab[i]);
    }
    printf("\n");
}

int main(void) {
    int tab[] = {1, 2, 3, 4, 5};
    int taille = sizeof(tab) / sizeof(tab[0]);

    printf("Tableau original : ");
    afficher(tab, taille);

    inverser(tab, taille);

    printf("Tableau inversé : ");
    afficher(tab, taille);

    return 0;
}
```

6.3 - Calculatrice améliorée

Reprendre la calculatrice de l'exercice 4 avec **switch**, en ajoutant :

- La gestion robuste de la division par zéro
- La possibilité de faire plusieurs calculs d'affilée (boucle)
- L'option de quitter le programme

Correction :

```
#include <stdio.h>

int main(void) {
    double a, b, resultat;
    char operation;
    char continuer;

    do {
        printf("\n=== Calculatrice ===\n");
        printf("Entrez deux nombres : ");
        scanf("%lf %lf", &a, &b);

        printf("Opération (+, -, *, /) : ");
        scanf(" %c", &operation);

        switch(operation) {
            case '+':
                resultat = a + b;
                printf("%.2f + %.2f = %.2f\n", a, b, resultat);
                break;

            case '-':
                resultat = a - b;
                printf("%.2f - %.2f = %.2f\n", a, b, resultat);
                break;
```

```
            case '*':
                resultat = a * b;
                printf("%.2f * %.2f = %.2f\n", a, b, resultat);
                break;

            case '/':
                if (b == 0 || b == -0.0) {
                    printf("Erreur : division par zéro !\n");
                } else {
                    resultat = a / b;
                    printf("%.2f / %.2f = %.2f\n", a, b, resultat);
                }
                break;

            default:
                printf("Opération invalide !\n");
        }

        printf("\nContinuer ? (o/n) : ");
        scanf(" %c", &continuer);

    } while (continuer == 'o' || continuer == 'O');

    printf("Au revoir !\n");
    return 0;
}
```

Exercice 7 : Matrice transposée

Écrire un programme qui lit une matrice 3×3 et affiche sa transposée.

Exercice 8 : Gestion d'un carnet d'adresses

Créer un programme utilisant un tableau de structures pour gérer un carnet d'adresses simple :

— Ajouter un contact (nom, téléphone, email)

— Afficher tous les contacts

— Rechercher un contact par nom

— Sauvegarder dans un fichier

Annexe A : Compilation et débogage

Compiler un programme

```
# Compilation simple
gcc programme.c -o programme

# Avec warnings (recommandé)
gcc -Wall -Wextra programme.c -o programme

# Avec bibliothèque math
gcc programme.c -o programme -lm

# Mode debug (avec gdb)
gcc -g programme.c -o programme
```

Erreurs de compilation fréquentes

Message d'erreur	Cause probable
`undefined reference to 'sqrt'`	Oubli de `-lm` avec math.h
`implicit declaration of function`	Oubli du `#include`
`expected ';' before ...`	Point-virgule manquant
`incompatible types`	Mauvais type de variable
`undeclared identifier`	Variable non déclarée

Erreurs à l'exécution

- **Segmentation fault** : accès mémoire invalide (pointeur NULL, tableau hors limites)
- **Bus error** : accès mémoire mal aligné
- **Stack overflow** : récursion trop profonde
- **Memory leak** : oubli de `free()`

Techniques de débogage simples

```c
// Méthode 1 : printf de débogage
printf("DEBUG: x = %d\n", x);

// Méthode 2 : Vérification d'assertions
#include <assert.h>
assert(x > 0);  // Arrête si faux

// Méthode 3 : Utiliser gdb
// $ gdb ./programme
// (gdb) run
// (gdb) backtrace
// (gdb) print variable
```

Annexe B : Fonctions de bibliothèque courantes

stdio.h - Entrées/sorties

```
printf, scanf, fprintf, fscanf
fgets, fputs, fgetc, fputc
fopen, fclose, fread, fwrite
feof, ferror, rewind, fseek
```

string.h - Manipulation de chaînes

```
strlen, strcpy, strncpy
strcat, strncat
strcmp, strncmp
strchr, strrchr, strstr
```

stdlib.h - Utilitaires

```
malloc, calloc, realloc, free
atoi, atof, atol
rand, srand
abs, labs
exit
```

math.h - Mathématiques

```
sqrt, pow, exp, log
sin, cos, tan
asin, acos, atan
ceil, floor, round
fabs
```

Note : Compiler avec `-lm`

ctype.h - Tests sur caractères

```
isalpha, isdigit, isalnum
isupper, islower
isspace, ispunct
toupper, tolower
```

Glossaire

Affectation Opération qui donne une valeur à une variable (`x = 5;`)
Argument Valeur passée à une fonction lors de son appel
Bibliothèque Ensemble de fonctions prêtes à l'emploi (`stdio.h`, `math.h`...)
Compilation Transformation du code source en code machine exécutable
Déclaration Annonce de l'existence d'une variable ou fonction
Définition Déclaration accompagnée de l'implémentation (pour une fonction)
Fichier en-tête Fichier `.h` contenant des déclarations
Fonction Bloc de code réutilisable avec un nom, des paramètres et un type de retour
Itération Une exécution du corps d'une boucle
Paramètre Variable dans la définition d'une fonction
Portée Zone du code où une variable est accessible
Prototype Déclaration d'une fonction (sans son corps)
Récursivité Fonction qui s'appelle elle-même
Segmentation fault Erreur due à un accès mémoire invalide
Type Catégorie d'une variable (`int`, `float`, `char`...)