Total /100

Identification

Prénom: .....

Nom: .....

Code permanent: .....

Instructions

- **Aucune documentation** n'est permise. Les rappels nécessaires sont donnés dans les questions.
- Avant de quitter, vous devez présenter votre carte étudiante et signer la feuille de présence.
- Pour chacune des questions à choix multiples, vous devez sélectionner **exactement une** réponse.
- Les questions 1 à 25 valent 80 points; les questions 26 et 27 valent 20 points.
- Pour chacune des questions, vous pouvez supposer que:
  - les programmes sont compilés avec `gcc` (norme C23) sous Linux 64 bits;
  - les en-têtes nécessaires (`<stdio.h>`, `<stdint.h>`, etc.) sont inclus;
  - `sizeof(char)` retourne 1, `sizeof(int)` retourne 4, `sizeof(double)` retourne 8.

Q1. /3 On lance la commande suivante dans un répertoire qui contient le fichier `calcul.c`:

```
$ gcc -Wall -std=c23 -o moyenne calcul.c
```

Si la compilation réussit, quel est le nom du fichier exécutable produit?

- ☐ `a.out`
- ☐ `calcul.o`
- ☐ `calcul`
- ■ `moyenne`

Q2. /4 Considérez le `Makefile` suivant:

```
prog: main.o calcul.o
  gcc -o prog main.o calcul.o

main.o: main.c
  gcc -std=c23 -c main.c

calcul.o: calcul.c
  gcc -std=c23 -c calcul.c

clean:
  rm -f prog *.o
```

On a lancé `make` une première fois, avec succès. On modifie ensuite *seulement* le fichier `calcul.c`, puis on relance `make`. Pour chacune des affirmations suivantes, indiquez si elle est vraie (V) ou fausse (F) en encerclant la lettre appropriée.
Rappel: Make n'exécute les commandes d'une règle que si la cible n'existe pas ou si l'une de ses dépendances est plus récente qu'elle.

(a) `main.c` est recompilé — V / (F)

(b) `calcul.c` est recompilé — (V) / F

(c) `prog` est produit de nouveau — (V) / F

(d) Les fichiers `.o` sont supprimés — V / (F)

Q3. /3 Considérez le fichier `notes.c` suivant et le message produit par sa compilation:

```
int main(void) {
  double moyenne = 72.5;
  printf("%d\n", moyenne);
  return 0;
}
```

```
$ gcc -Wall -Wextra -std=c23 -o notes notes.c
notes.c:3:12: warning: format '%d' expects argument
    of type 'int', but argument 2 has type 'double
    ' [-Wformat=]
```

Que peut-on conclure?

- ☐ La compilation échoue: aucun exécutable n'est produit
- ☐ Un exécutable est produit, et il affichera `72`
- ☐ Un exécutable est produit, et il affichera `72.5`
- ■ Un exécutable est produit, mais son affichage est imprévisible

Q4. /2 Un capteur produit des mesures entières comprises entre 0 et 1000. Quel type de `<stdint.h>` permet de stocker une mesure en utilisant le moins de mémoire possible?
Rappel: le type `uintN_t` représente les entiers non signés de N bits, de 0 à $2^N - 1$, et le type `intN_t` les entiers signés de N bits, de -2^{N-1} à $2^{N-1} - 1$.

- ☐ `int8_t`
- ☐ `uint8_t`
- ■ `uint16_t`
- ☐ `uint32_t`

Q5. /4 Dans cette question, on considère des entiers signés sur 8 bits (type `int8_t`), représentés en complément à deux.
Rappel: en complément à deux, le bit de poids fort vaut 0 pour une valeur positive et 1 pour une valeur négative; on obtient la représentation de $-x$ en inversant tous les bits de la représentation de x, puis en ajoutant 1.

(a) Donnez la représentation hexadécimale (sur 8 bits) de chacune des valeurs suivantes:

45: ..... 0x2D .....

-6: ..... 0xFA .....

(b) Donnez la valeur décimale représentée par chacun des octets suivants:

`0x5A`: ..... 90 .....

`0x9C`: ..... -100 .....

Q6. /2 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
char c = 'C';
printf("%c\n", c + 2);
printf("%d\n", c - 'A');
```

Rappel: en ASCII, le caractère `'A'` a pour code 65, et les lettres majuscules ont des codes consécutifs, dans l'ordre alphabétique.

(a) Ligne 1: ..... E .....
(b) Ligne 2: ..... 2 .....

Q7. /4 Considérez le fragment de code suivant:

```
int a = 0, b = 0;
int k = scanf("%d %d", &a, &b);
printf("%d %d %d\n", k, a, b);
```

Rappel: `scanf` lit les valeurs dans l'ordre, s'arrête dès qu'une entrée ne correspond pas au format attendu, et retourne le nombre d'éléments lus avec succès.
Qu'est-ce qui sera affiché sur la sortie standard si on entre...

(a) ...`12 abc`: ..... 1 12 0 .....
(b) ...`abc 12`: ..... 0 0 0 .....

Q8. /4 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
int a = 17, b = 5;
printf("%d\n", a / b);
printf("%d\n", a % b);
printf("%.2f\n", (a / b) * 1.0);
printf("%.2f\n", (double) a / b);
```

(a) Ligne 1: ..... 3 .....
(b) Ligne 2: ..... 2 .....
(c) Ligne 3: ..... 3.00 .....
(d) Ligne 4: ..... 3.40 .....

Q9. /3 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
unsigned int u = 3;
int i = -1;
if (i < u)
  printf("A\n");
else
  printf("B\n");
```

Rappel: dans une expression où apparaissent à la fois un `int` et un `unsigned int`, l'`int` est d'abord converti en `unsigned int`.

- ☐ `A`, car $-1 < 3$
- ■ `B`, car -1 est converti en un très grand entier positif
- ☐ `B`, car -1 est converti en 1
- ☐ Rien: la comparaison provoque une erreur de compilation

Q10. /4 On considère la variable suivante:

```
uint8_t x = 0x5A;    // 0101 1010
```

Donnez la valeur, en hexadécimal, de chacune des expressions suivantes. Les expressions sont *indépendantes*: aucune ne modifie `x`.
Rappel: `|` (ou bit à bit), `&` (et bit à bit), `^` (ou exclusif bit à bit), `>>` (décalage vers la droite).

(a) `x | 0x01`: ..... 0x5B .....
(b) `x & 0x0F`: ..... 0x0A .....
(c) `x ^ 0xFF`: ..... 0xA5 .....
(d) `x >> 2`: ..... 0x16 .....

Q11. /2 Qu'est-ce qui sera affiché sur la sortie standard par chacun des fragments de code suivants?

(a)
```
int a = 5;
int b = a++;
printf("%d %d\n", a, b);
```
Réponse: ..... 6 5 .....

(b)
```
int a = 5;
int b = ++a;
printf("%d %d\n", a, b);
```
Réponse: ..... 6 6 .....

Q12. /2 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
uint8_t c = 250;
c += 10;
printf("%d\n", c);
c -= 5;
printf("%d\n", c);
```

Rappel: le type `uint8_t` représente les entiers non signés de 8 bits, de 0 à 255.

(a) Ligne 1: ..... 4 .....
(b) Ligne 2: ..... 255 .....

Q13. /3 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
int total = 0;
for (int i = 0; i < 4; i++) {
  switch (i) {
    case 0:  total += 1; break;
    case 1:
    case 2:  total += 10;
    default: total += 100;
  }
}
printf("%d\n", total);
```

Réponse: ..... 321 .....

Q14. /4 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
int s = 0;
for (int i = 1; i <= 10; i++) {
  if (i % 3 == 0) continue;
  if (i > 7) break;
  s += i % 2 == 0 ? i : -i;
}
printf("%d\n", s);
```

Rappel: `c ? x : y` vaut `x` si `c` est vraie, et `y` sinon.
Réponse: ..... -7 .....

Q15. /4 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
int n = 4, k = 0;
while (n > 5) {
  n /= 2;
  k++;
}
printf("%d %d\n", n, k);
n = 4;
k = 0;
do {
  n /= 2;
  k++;
} while (n > 5);
printf("%d %d\n", n, k);
```

(a) Ligne 1: ..... 4 0 .....
(b) Ligne 2: ..... 2 1 .....

Q16. /3 Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant? *Suggestion:* faites une trace à la main.

```
int total = 0;
for (int i = 1; i <= 4; i++)
  for (int j = i; j <= 4; j += 2)
    total += j;
printf("%d\n", total);
```

Réponse: ..... 17 .....

Q17. /3 Un capteur de température est considéré valide si sa mesure `t` (un entier) satisfait $-40 \leq t \leq 85$. Une fonction `int est_valide(int t)` doit retourner 1 dans ce cas, et 0 sinon. Quel ensemble d'entrées permet le mieux de tester les *cas limites* de cette fonction?

- ■ $-41, -40, 85, 86$
- ☐ $-100, 0, 100$
- ☐ $-40, 0, 85$
- ☐ $-30, 20, 70, 80$

Q18. /4 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
int x = 1;

void f(void) {
  static int s = 0;
  int a = 0;
  s += x;
  a += x;
  x++;
  printf("%d %d\n", s, a);
}

int main(void) {
  f();
  int x = 10;
  f();
  {
    int x = 100;
    f();
  }
  printf("%d\n", x);
  return 0;
}
```

(a) Ligne 1: ..... 1 1 .....
(b) Ligne 2: ..... 3 2 .....
(c) Ligne 3: ..... 6 3 .....
(d) Ligne 4: ..... 10 .....

Q19. /2 (Suite de la question précédente) Dans quelle zone de la mémoire se trouve la variable `s` de la fonction `f`?

- ☐ La pile
- ☐ Le tas
- ■ Les données statiques
- ☐ Le code

Q20. /4 Pour chacune des affirmations suivantes, indiquez si elle est vraie (V) ou fausse (F) en encerclant la lettre appropriée.

(a) Une variable locale déclarée sans initialisation vaut `0` — V / (F)
(b) Une variable globale déclarée sans initialisation vaut `0` — (V) / F
(c) Une variable locale `static` est visible dans toutes les fonctions du fichier — V / (F)
(d) La portée d'une variable est déterminée à la compilation, sa durée de vie concerne l'exécution — (V) / F

Q21. /4 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
int carre(int x) {
  x = x * x;
  return x;
}

double moyenne(int a, int b) {
  return (a + b) / 2;
}

int signe(double x) {
  if (x > 0) return 1;
  if (x < 0) return -1;
  return 0;
}

int main(void) {
  int x = 3;
  int y = carre(x);
  printf("%d %d\n", x, y);
  printf("%.1f\n", moyenne(3, 4));
  printf("%d\n", signe(-0.5));
  printf("%d\n", carre(signe(2) + 1));
  return 0;
}
```

(a) Ligne 1: ..... 3 9 .....
(b) Ligne 2: ..... 3.0 .....
(c) Ligne 3: ..... -1 .....
(d) Ligne 4: ..... 4 .....

Q22. /3 La compilation du programme suivant produit un diagnostic à la ligne où `carre` est appelée.

```
int main(void) {
  printf("%.2f\n", carre(1.5));
  return 0;
}

double carre(double x) {
  return x * x;
}
```

Quelle modification corrige le problème, sans déplacer la définition de `carre`?

- ■ Ajouter `double carre(double x);` avant `main`
- ☐ Ajouter `double carre(double x);` après `main`
- ☐ Ajouter `#include <math.h>` en haut du fichier
- ☐ Remplacer `carre(1.5)` par `carre((double) 1.5)`

Q23. /4 Considérez la fonction récursive suivante:

```
int g(int n) {
  if (n <= 1) return n;
  if (n % 2 == 0) return g(n / 2) + 1;
  return g(n - 1) + 2;
}
```

Complétez le tableau suivant:

n	0	2	6	7
g(n)	0	2	5	7

Q24. /2 (Suite de la question précédente) Combien d'appels à la fonction `g` sont effectués au total lors de l'évaluation de `g(7)`, en comptant l'appel `g(7)` lui-même?

☐ 3 ☐ 4 ■ 5 ☐ 7

Q25. /3 Qu'est-ce qui sera affiché sur la sortie standard par l'appel `r(3)`?

```
void r(int n) {
  if (n == 0) return;
  r(n - 1);
  printf("%d ", n);
  r(n - 1);
}
```

Réponse: ..... 1 2 1 3 1 2 1 .....

Q26. /10 On souhaite compter le nombre de bits qui valent 1 dans un entier non signé de 32 bits. Par exemple, `0xA5` $= (1010\,0101)_2$ contient 4 bits valant 1.
Rappel:

- `x & 1` vaut le bit de poids faible de `x` (`0` ou `1`);
- `x >> 1` décale les bits de `x` d'une position vers la droite: le bit de poids faible est perdu et un `0` est inséré à gauche.

(a) **(6 points)** Écrivez la définition de la fonction suivante, documentée par sa *docstring*, en utilisant une *boucle*:

```
/**
 * Compte le nombre de bits valant 1 dans un entier non signe.
 *
 * @param x  L'entier dont on compte les bits
 * @return   Le nombre de bits de x qui valent 1
 */
int nombre_bits_actifs(uint32_t x);
```

La fonction ne doit rien afficher.

(b) **(4 points)** Proposez quatre cas de test pour la fonction `nombre_bits_actifs`, chacun formé d'une entrée et du résultat attendu. Au moins deux d'entre eux doivent être des *cas limites*. Justifiez brièvement le choix de chacun.

Solution (a)

```
int nombre_bits_actifs(uint32_t x) {
  int nb = 0;
  while (x != 0) {
    nb += x & 1;
    x >>= 1;
  }
  return nb;
}
```

Barème suggéré: extraction correcte de chaque bit avec `&` et `>>` (2 points); boucle correcte qui se termine (2 points); résultat retourné et non affiché (1 point); lisibilité (1 point). Une boucle `for` sur les 32 positions (`(x >> i) & 1`) est aussi acceptée.

Solution (b) Par exemple:

- `0` → `0`: cas limite, aucun bit à 1 (la boucle ne s'exécute pas);
- `1` → `1`: cas limite, une seule itération;
- `0xFFFFFFFF` → `32`: cas limite, tous les bits à 1 (valeur maximale);
- `0x80000000` → `1`: cas limite, seul le bit de poids fort est à 1;
- `0xA5` → `4`: cas normal.

Barème suggéré: 1 point par cas pertinent, avec résultat attendu correct et justification; au plus 2 points si moins de deux cas limites.

Q27. /10 Un microcontrôleur lit un signal numérique qui vaut `0` ou `1`. On appelle *front montant* le passage du signal de la valeur `0` à la valeur `1` entre deux lectures consécutives. On suppose que le signal vaut `0` avant la première lecture. Par exemple, la suite de lectures `1 1 0 1 0 0 1 1` contient 3 fronts montants (aux 1re, 4e et 7e lectures).
Rappel:

- une variable locale déclarée `static` est initialisée une seule fois et conserve sa valeur d'un appel à l'autre;
- `scanf("%d", &n)` retourne le nombre d'éléments lus avec succès (`1` si un entier a été lu dans `n`).

(a) **(5 points)** Écrivez la définition de la fonction suivante, documentée par sa *docstring*:

```
/**
 * Indique si le signal vient de passer de 0 a 1.
 *
 * Le signal est suppose valoir 0 avant le premier appel.
 *
 * @param signal  La valeur courante du signal (0 ou 1)
 * @return        1 si le signal valait 0 a l'appel precedent
 *                et vaut 1 maintenant, 0 sinon
 */
int front_montant(int signal);
```

Par exemple, le fragment de code suivant affiche les valeurs indiquées en commentaire:

```
// Ici, front_montant n'a encore jamais ete appelee
printf("%d\n", front_montant(1));  // 1 (passage de 0 a 1)
printf("%d\n", front_montant(1));  // 0
printf("%d\n", front_montant(0));  // 0
printf("%d\n", front_montant(0));  // 0
printf("%d\n", front_montant(1));  // 1 (passage de 0 a 1)
```

Vous ne devez *pas* utiliser de variable globale.

(b) **(5 points)** Écrivez la fonction `main` d'un programme qui lit sur l'entrée standard une suite d'entiers (`0` ou `1`), jusqu'à ce que la lecture échoue (par exemple, à la fin de l'entrée), puis qui affiche sur la sortie standard le nombre de fronts montants, suivi d'un saut de ligne. Votre programme doit utiliser la fonction `front_montant`. Par exemple:

```
$ echo "1 1 0 1 0 0 1 1" | ./fronts
3
```

Solution (a)

```
int front_montant(int signal) {
  static int precedent = 0;
  int front = precedent == 0 && signal == 1;
  precedent = signal;
  return front;
}
```

Barème suggéré: variable locale `static` initialisée à `0` (2 points); valeur retournée correcte (1,5 point); mise à jour de l'état à chaque appel (1,5 point). L'utilisation d'une variable globale entraîne la perte des points associés à `static`.

Solution (b)

```
int main(void) {
  int signal;
  int nb_fronts = 0;
  while (scanf("%d", &signal) == 1)
    nb_fronts += front_montant(signal);
  printf("%d\n", nb_fronts);
  return 0;
}
```

Barème suggéré: boucle de lecture contrôlée par la valeur de retour de `scanf` (2,5 points, dont 0,5 pour le `&`); utilisation de `front_montant` (1,5 point); affichage final et `return 0` (1 point).