Total /100

Identification

Prénom: .....

Nom: .....

Code permanent: .....

Instructions

- Cet examen de **pratique** a la même forme que l'examen intra: **aucune documentation** n'est permise, et les rappels nécessaires sont donnés dans les questions.
- 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 `capteur.c`:

```
$ gcc -Wall -std=c23 -c capteur.c
```

Rappel: l'option `-c` demande à `gcc` de s'arrêter avant l'édition des liens.
Si la compilation réussit, quel fichier est produit?

- ☐ `a.out`
- ■ `capteur.o`
- ☐ `capteur`
- ☐ `capteur.h`

Q2. /4 Considérez le `Makefile` suivant:

```
jeu: jeu.o carte.o
  gcc -o jeu jeu.o carte.o

jeu.o: jeu.c carte.h
  gcc -std=c23 -c jeu.c

carte.o: carte.c carte.h
  gcc -std=c23 -c carte.c
```

On a lancé `make` une première fois, avec succès. On modifie ensuite *seulement* le fichier `jeu.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) `jeu.c` est recompilé — (V) / F
(b) `carte.c` est recompilé — V / (F)
(c) `jeu` est produit de nouveau — (V) / F
(d) Si on relance `make` immédiatement après, aucune commande n'est exécutée — (V) / F

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

```
int main(void) {
  for (int i = 0; i < 3; i++) {
    int total = i * 2;
  }
  printf("%d\n", total);
  return 0;
}
```

```
$ gcc -Wall -std=c23 -o portee portee.c
portee.c:5:18: error: 'total' undeclared (first use
    in this function)
```

Que peut-on conclure?

- ■ Aucun exécutable n'est produit, car `total` n'est pas visible à la ligne 5
- ☐ Un exécutable est produit, et il affichera `4`
- ☐ Un exécutable est produit, et il affichera `0`
- ☐ Un exécutable est produit, mais son affichage est imprévisible

Q4. /2 Un accéléromètre produit des mesures entières comprises entre -500 et 500. 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`
- ■ `int16_t`
- ☐ `uint8_t`
- ☐ `uint16_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:

100: ..... 0x64 .....

-20: ..... 0xEC .....

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

`0x3F`: ..... 63 .....

`0xF0`: ..... -16 .....

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

```
char c = 'f';
printf("%d\n", c - 'a');
printf("%c\n", 'a' + 10);
```

Rappel: en ASCII, le caractère `'a'` a pour code 97, et les lettres minuscules ont des codes consécutifs, dans l'ordre alphabétique.

(a) Ligne 1: ..... 5 .....
(b) Ligne 2: ..... k .....

Q7. /4 Considérez le fragment de code suivant:

```
int n = 7;
double x = 1.5;
int k = scanf("%d %lf", &n, &x);
printf("%d %d %.1f\n", k, n, x);
```

Rappel: `%lf` lit un `double` avec `scanf`; `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) ...`3 2.5`: ..... 2 3 2.5 .....
(b) ...`4 xyz`: ..... 1 4 1.5 .....

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

```
int a = 23, b = 4;
printf("%d\n", a / b);
printf("%d\n", a % b);
printf("%.2f\n", (double) (a / b));
printf("%.2f\n", a / (double) b);
```

(a) Ligne 1: ..... 5 .....
(b) Ligne 2: ..... 3 .....
(c) Ligne 3: ..... 5.00 .....
(d) Ligne 4: ..... 5.75 .....

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

```
int n = 7;
double moyenne = n / 2;
int t = 9.99;
printf("%.1f %d\n", moyenne, t);
```

- ☐ `3.5 9`
- ■ `3.0 9`
- ☐ `3.5 10`
- ☐ `3.0 10`

Q10. /4 On considère la variable suivante:

```
uint8_t x = 0xC3;    // 1100 0011
```

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 & 0xF0`: ..... 0xC0 .....
(b) `x | 0x0C`: ..... 0xCF .....
(c) `x ^ 0x0F`: ..... 0xCC .....
(d) `x >> 4`: ..... 0x0C .....

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

(a)
```
int a = 3;
int b = 2 * a++;
printf("%d %d\n", a, b);
```
Réponse: ..... 4 6 .....

(b)
```
int a = 3;
int b = 2 * ++a;
printf("%d %d\n", a, b);
```
Réponse: ..... 4 8 .....

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

```
uint8_t v = 5;
v -= 10;
printf("%d\n", v);
v += 20;
printf("%d\n", v);
```

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

(a) Ligne 1: ..... 251 .....
(b) Ligne 2: ..... 15 .....

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

```
int total = 0;
for (int i = 1; i <= 4; i++) {
  switch (i % 3) {
    case 0:  total += 100; break;
    case 1:  total += 10;
    case 2:  total += 1; break;
    default: total += 1000;
  }
}
printf("%d\n", total);
```

Réponse: ..... 123 .....

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

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

Rappel: `c ? x : y` vaut `x` si `c` est vraie, et `y` sinon.
Réponse: ..... 12 .....

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

```
int n = 1, k = 0;
while (n < 1) {
  n *= 3;
  k++;
}
printf("%d %d\n", n, k);
n = 1;
k = 0;
do {
  n *= 3;
  k++;
} while (n < 20);
printf("%d %d\n", n, k);
```

(a) Ligne 1: ..... 1 0 .....
(b) Ligne 2: ..... 27 3 .....

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 <= 3; i++)
  for (int j = 1; j <= i; j++)
    total += i * j;
printf("%d\n", total);
```

Réponse: ..... 25 .....

Q17. /3 Une fonction `int categorie(int age)` retourne 0 si `age` < 12, 1 si $12 \leq$ `age` < 18, et 2 sinon. Quel ensemble d'entrées permet le mieux de tester les *cas limites* de cette fonction?

- ☐ 5, 15, 30
- ■ 11, 12, 17, 18
- ☐ 0, 12, 18
- ☐ 11, 18, 100

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

```
int compteur = 0;

int suivant(void) {
  static int n = 10;
  int k = 1;
  n += k;
  k++;
  compteur++;
  return n;
}

int main(void) {
  printf("%d\n", suivant());
  int compteur = 5;
  printf("%d\n", suivant());
  {
    int compteur = 50;
    printf("%d\n", suivant() + compteur);
  }
  printf("%d\n", compteur);
  return 0;
}
```

(a) Ligne 1: ..... 11 .....
(b) Ligne 2: ..... 12 .....
(c) Ligne 3: ..... 63 .....
(d) Ligne 4: ..... 5 .....

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

- ■ 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) Deux fonctions différentes peuvent chacune déclarer une variable locale nommée `n`, sans conflit — (V) / F

(b) Une variable déclarée dans l'en-tête d'une boucle `for` reste accessible après la boucle — V / (F)

(c) Une variable locale `static` est réinitialisée à chaque appel de sa fonction — V / (F)

(d) Les variables locales sans `static` sont stockées sur la pile — (V) / F

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

```
int plus_un(int n) {
  n++;
  return n;
}

double demi(int n) {
  return n / 2.0;
}

int max3(int a, int b, int c) {
  int m = a;
  if (b > m) m = b;
  if (c > m) m = c;
  return m;
}

int main(void) {
  int n = 5;
  int r = plus_un(n);
  printf("%d %d\n", n, r);
  printf("%.1f\n", demi(7));
  printf("%d\n", max3(4, 9, 2));
  printf("%d\n", plus_un(max3(1, 2, 3)));
  return 0;
}
```

(a) Ligne 1: ..... 5 6 .....
(b) Ligne 2: ..... 3.5 .....
(c) Ligne 3: ..... 9 .....
(d) Ligne 4: ..... 4 .....

Q22. /3 Considérez les deux extraits suivants, tirés d'un même fichier:

```
double aire(double rayon);
```

```
double aire(double rayon) {
  return 3.14159 * rayon * rayon;
}
```

Laquelle des affirmations suivantes est vraie?

- ■ Le premier extrait est un prototype: il se termine par ; et n'a pas de corps
- ☐ Le premier extrait doit obligatoirement être placé après le second
- ☐ Le nom `rayon` est obligatoire dans le premier extrait
- ☐ Le premier extrait réserve de la mémoire pour le paramètre `rayon`

Q23. /4 Considérez la fonction récursive suivante:

```
int h(int n) {
  if (n < 10) return n;
  return h(n / 10) + n % 10;
}
```

Complétez le tableau suivant:

n	7	42	305	999
h(n)	7	6	8	27

Q24. /2 (Suite de la question précédente) Combien d'appels à la fonction `h` sont effectués au total lors de l'évaluation de `h(305)`, en comptant l'appel `h(305)` lui-même?

☐ 1 ☐ 2 ■ 3 ☐ 4

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

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

Réponse: ..... 3 2 1 1 2 3 .....

Q26. /10 Un entier n est *premier* s'il est supérieur ou égal à 2 et si ses seuls diviseurs positifs sont 1 et n. Par exemple, 2, 3, 5 et 97 sont premiers, alors que 0, 1, 4 et 9 ne le sont pas.
Rappel: l'entier `d` divise l'entier `n` si et seulement si `n % d == 0`.

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

```
/**
 * Indique si un entier est premier.
 *
 * L'entier n doit etre positif ou nul.
 *
 * @param n  L'entier a verifier
 * @return   1 si n est premier, 0 sinon
 */
int est_premier(int n);
```

La fonction ne doit rien afficher.

(b) **(4 points)** Proposez quatre cas de test pour la fonction `est_premier`, 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 est_premier(int n) {
  if (n < 2) return 0;
  for (int d = 2; d * d <= n; d++)
    if (n % d == 0)
      return 0;
  return 1;
}
```

Barème suggéré: traitement de 0 et 1 (1,5 point); boucle qui teste les diviseurs possibles et se termine (2,5 points); résultat retourné et non affiché (1 point); lisibilité (1 point). Une boucle qui s'arrête à `d < n` plutôt qu'à `d * d <= n` est aussi acceptée: elle est moins efficace, mais correcte.

Solution (b) Par exemple:

- `0` → `0` et `1` → `0`: cas limites, sous le seuil de 2;
- `2` → `1`: cas limite, le plus petit nombre premier (la boucle ne s'exécute pas);
- `4` → `0`: cas limite, le plus petit nombre non premier supérieur à 1;
- `25` → `0`: cas limite, carré d'un nombre premier (vérifie la borne de la boucle);
- `97` → `1`: 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 Une station météo reçoit une suite de mesures de température (des entiers). On dit qu'une mesure est un *record* si elle est strictement supérieure à toutes les mesures précédentes. Par convention, la première mesure est toujours un record. Par exemple, la suite `3 1 4 1 5 9 2 6` contient 4 records: `3`, `4`, `5` et `9`.
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 une mesure est un record.
 *
 * Une mesure est un record si elle est strictement superieure a
 * toutes les mesures recues lors des appels precedents. La mesure
 * recue lors du premier appel est toujours un record.
 *
 * @param mesure  La nouvelle mesure
 * @return        1 si la mesure est un record, 0 sinon
 */
int est_record(int mesure);
```

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

```
// Ici, est_record n'a encore jamais ete appelee
printf("%d\n", est_record(-5));  // 1 (premiere mesure)
printf("%d\n", est_record(-8));  // 0
printf("%d\n", est_record(-5));  // 0 (egale, pas superieure)
printf("%d\n", est_record(2));   // 1
printf("%d\n", est_record(1));   // 0
```

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, 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 records, suivi d'un saut de ligne. Votre programme doit utiliser la fonction `est_record`. Par exemple:

```
$ echo "3 1 4 1 5 9 2 6" | ./records
4
```

Solution (a)

```
int est_record(int mesure) {
  static int premier_appel = 1;
  static int maximum = 0;
  if (premier_appel || mesure > maximum) {
    premier_appel = 0;
    maximum = mesure;
    return 1;
  }
  return 0;
}
```

Barème suggéré: variables locales `static` pour mémoriser le maximum (2 points); traitement correct du premier appel (1,5 point); comparaison stricte et mise à jour du maximum (1,5 point). L'utilisation d'une variable globale entraîne la perte des points associés à `static`. Initialiser `maximum` à une très petite valeur (par exemple `INT_MIN`, de `<limits.h>`) sans traiter le premier appel est accepté, à condition que la mesure `INT_MIN` elle-même soit considérée comme impossible.

Solution (b)

```
int main(void) {
  int mesure;
  int nb_records = 0;
  while (scanf("%d", &mesure) == 1)
    nb_records += est_record(mesure);
  printf("%d\n", nb_records);
  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 `est_record` (1,5 point); affichage final et `return 0` (1 point).