Identification /5

Prénom: .....

Nom: .....

Code permanent: .....

Veuillez choisir **une seule réponse** par question, avec une croix (×) ou un crochet (✓).

Q1. Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
int a = 7, b = 2;
double x = a / b;
double y = (double) a / b;
printf("%.1f %.1f %d\n", x, y, a % b);
```

- ☐ `3.5 3.5 1`
- ☐ `3.0 3.0 1`
- ■ `3.0 3.5 1`
- ☐ `3.0 3.5 3`

Q2. Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
unsigned char port = 0x0F;
port |= 1 << 5;
port &= 0xF3;
printf("%d\n", port);
```

- ☐ `3`
- ☐ `47`
- ■ `35`
- ☐ `255`

Q3. Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

```
signed char c = -1;
unsigned char u = (unsigned char) c;
printf("%d %d\n", c, u);
```

- ☐ `-1 -1`
- ■ `-1 255`
- ☐ `-1 1`
- ☐ `255 255`

Q4. Qu'est-ce qui sera affiché sur la sortie standard par le fragment de code suivant?

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

- ☐ `10`
- ☐ `111`
- ■ `110`
- ☐ `1110`

Q5. Écrivez un programme C complet (avec `#include` et `main`) qui lit sur l'entrée standard une température `t` en degrés Celsius, de type `double`. Si la lecture échoue, le programme doit afficher un message d'erreur sur la *sortie d'erreur* et se terminer avec le code de retour 1. Sinon, il affiche, suivi d'un saut de ligne:

- `gel` si `t` est strictement inférieure à 0;
- `normal` si `t` est entre 0 et 30 inclusivement;
- `alarme` si `t` est strictement supérieure à 30.

Par exemple, si l'entrée est `12.5`, alors le programme affiche `normal`.
Rappel:

`%lf`	`double` avec `scanf`
`k = scanf(...)`	`k`: nombre d'éléments lus
`fprintf(stderr, ...)`	écrit sur la sortie d'erreur

Note: utilisez le verso au besoin.

Voir solution page suivante

Q1. Comme `a` et `b` sont de type `int`, l'expression `a / b` est une *division entière*, qui donne `3`: ce n'est qu'*après* le calcul que la valeur est convertie en `double` lors de l'affectation, d'où `x` = 3,0. Dans `(double) a / b`, le *cast* est prioritaire sur la division: `a` est converti en `double` *avant* le calcul, puis `b` est converti implicitement, ce qui donne une division flottante et `y` = 3,5. Enfin, `a % b` est le reste de la division entière de 7 par 2, soit `1` (et non le quotient `3`). La sortie est donc `3.0 3.5 1`. Les réponses `3.5 3.5 1`, `3.0 3.0 1` et `3.0 3.5 3` proviennent respectivement d'une division flottante dans `a / b`, d'un *cast* appliqué après la division et d'une confusion entre `%` et `/`.

Q2. Au départ, `port` vaut `0x0F` = `0000 1111`. Le masque `1 << 5` vaut `0010 0000`, donc `port |= 1 << 5` *active* le bit 5: on obtient `0010 1111` = `0x2F` = 47. Ensuite, `0xF3` = `1111 0011`, donc `port &= 0xF3` *désactive* les bits 2 et 3 en conservant les autres: on obtient `0010 0011` = `0x23`. Le spécificateur `%d` affiche cette valeur en décimal: $2 \times 16 + 3 = 35$. Les réponses `47`, `3` et `255` (`0x2F`, `0x03` et `0xFF`) correspondent respectivement à l'oubli de la désactivation, à l'oubli de l'activation et à une confusion entre `&` et `|`.

Q3. Sur 8 bits, en complément à deux, la valeur -1 de `c` est représentée par `1111 1111` = `0xFF`. La conversion en `unsigned char` conserve ces mêmes 8 bits, mais les interprète maintenant en binaire direct, sans signe: `u` vaut donc 255. Lors de l'appel à `printf`, les deux variables sont promues en `int` en conservant leur valeur, et `%d` affiche `-1` et `255`. Les réponses `-1 1`, `-1 -1` et `255 255` correspondent respectivement à l'idée erronée qu'on retire simplement le signe (valeur absolue), à l'oubli de la conversion et à l'idée erronée que le *cast* modifie aussi la variable `c`.

Q4. Puisque `n` vaut `2`, l'exécution saute directement à l'étiquette `case 2`, qui ajoute `10`. Comme il n'y a pas de `break` à la fin de ce cas, l'exécution *se poursuit* dans le cas suivant (`case 3`), qui ajoute `100`, puis le `break` fait sortir du `switch`. Le `case 1` n'est jamais exécuté, car il précède l'étiquette choisie, et le `default` non plus, car le `break` survient avant. On obtient donc $10 + 100 = 110$.

Q5. On lit la température avec `%lf` (et non `%f`, réservé à `printf`) et on vérifie que `scanf` retourne bien 1. En cas d'échec, on écrit le message sur la sortie d'erreur avec `fprintf(stderr, ...)` (et non avec `printf`, qui écrit sur la sortie standard), puis on retourne `1`. Pour la classification, une chaîne `if`/`else if`/`else` est tout indiquée, puisque les trois cas sont mutuellement exclusifs. Comme la condition d'un `else if` n'est évaluée que si les précédentes sont fausses, il est inutile de tester `t >= 0` dans la deuxième condition: on le sait déjà. De même, le `else` final n'a besoin d'aucune condition. Cela donne le programme suivant:

```
#include <stdio.h>

int main(void) {
  double t;
  if (scanf("%lf", &t) != 1) {
    fprintf(stderr, "erreur: lecture invalide\n");
    return 1;
  }
  if (t < 0) {
    printf("gel\n");
  } else if (t <= 30) {
    printf("normal\n");
  } else {
    printf("alarme\n");
  }
  return 0;
}
```