Identification /5

Prénom: .....

Nom: .....

Code permanent: .....

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

Q1. Quel type de la bibliothèque `stdint.h` représente un entier signé d'exactement 32 bits, peu importe la plateforme?

- ☐ `int`
- ■ `int32_t`
- ☐ `long`
- ☐ `long long`

Q2. Sur 8 bits, en complément à deux, quelle est la représentation hexadécimale de la valeur -1?

- ☐ `0x01`
- ☐ `0xFE`
- ☐ `0x80`
- ■ `0xFF`

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

```
#include <stdio.h>

int main(void) {
  printf("%d\n", 0x1F + 010 + 0b11);
  return 0;
}
```

- ■ `42`
- ☐ `44`
- ☐ `50`
- ☐ `52`

Q4. Qu'est-ce qui sera affiché sur la sortie standard par le programme suivant? *Rappel:* en ASCII, le caractère `'A'` a pour code 65.

```
#include <stdio.h>

int main(void) {
  char c = 'A' + 10;
  printf("%c %d\n", c, c);
  return 0;
}
```

- ☐ `J 74`
- ☐ `K 10`
- ■ `K 75`
- ☐ `75 K`

Q5. Écrivez un programme C complet (avec `#include` et `main`) qui lit sur l'entrée standard un entier `n` puis un nombre réel `x` de type `double`, séparés par un espace. Le programme doit ensuite afficher sur une seule ligne, suivie d'un saut de ligne, l'entier `n` aligné à droite sur 5 positions, puis le réel `x` avec 2 décimales sur 8 positions, chacun entre crochets. Par exemple, si l'entrée est `42 3.14159`, alors le programme affiche

```
[   42][    3.14]
```

Si la lecture échoue, le programme doit se terminer avec un code de retour différent de 0.
Rappel:

`%d`	`int` (`printf` et `scanf`)
`%f`	`double` avec `printf`
`%lf`	`double` avec `scanf`
`%5d`	largeur 5, alignement à droite
`%8.2f`	largeur 8, 2 décimales

Note: utilisez le verso au besoin.

Voir solution page suivante

Q1. Les types `int` et `long` ont une taille qui dépend de la plateforme (typiquement 4 et 8 octets sous Linux 64 bits), tout comme `long long`. Les types de `stdint.h` ont au contraire une taille fixe: `int32_t` est toujours un entier signé de 32 bits, ce qui est essentiel, par exemple, pour manipuler des registres matériels, des protocoles ou des fichiers binaires.

Q2. En complément à deux, on obtient l'opposé d'un entier en inversant tous les bits, puis en ajoutant 1. Sur 8 bits, on a $1 =$ `0000 0001`, dont l'inversion donne `1111 1110`, et l'ajout de 1 donne `1111 1111`, soit `0xFF`. On peut aussi le retenir ainsi: -1 est la valeur dont tous les bits valent 1.

Q3. Les préfixes déterminent la base des constantes littérales entières: `0x1F` est en hexadécimal ($1\times16+15 = 31$), `010` est en octal, à cause du `0` initial ($1 \times 8 = 8$) et `0b11` est en binaire (C23), ce qui donne 3. La somme vaut donc $31 + 8 + 3 = 42$. Les réponses `44`, `50` et `52` proviennent respectivement d'une lecture de `010` comme un décimal, de `0b11` comme un décimal et des deux erreurs simultanées.

Q4. Le type `char` est en réalité un petit entier et, en ASCII, `'A'` vaut 65. La variable `c` contient donc $65 + 10 = 75$, qui est le code du caractère `'K'`. Le spécificateur `%c` affiche le caractère (`K`) et `%d` affiche le code en décimal (`75`). La sortie est donc `K 75`.

Q5. Il faut lire un `int` avec `%d` et un `double` avec `%lf` (et non `%f`, qui est réservé à `printf`), en passant l'*adresse* des variables avec `&`. Comme `scanf` retourne le nombre d'éléments lus avec succès, on vérifie que la valeur retournée est bien 2. Pour l'affichage, `%5d` aligne l'entier à droite sur 5 positions et `%8.2f` affiche le réel avec 2 décimales sur 8 positions. Cela donne le programme suivant:

```
#include <stdio.h>

int main(void) {
  int n;
  double x;
  if (scanf("%d %lf", &n, &x) != 2)
    return 1;
  printf("[%5d][%8.2f]\n", n, x);
  return 0;
}
```