Total /100

Identification

Prénom: .....

Nom: .....

Code permanent: .....

Instructions

- Vous avez droit à une feuille de notes recto-verso manuscrite, au format lettre (8 1/2×11)
- 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
- Pour chacune des questions, vous pouvez supposer que:
  - `sizeof(int)` retourne 4;
  - `sizeof(char)` retourne 1;
  - `sizeof(bool)` retourne 1;
  - `sizeof(double)` retourne 8 et
  - `sizeof(T*)` retourne 8, peu importe `T`.

Q1. /3 Considérez le programme `prog.c` suivant:

```
#include <stdio.h>

int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; ++i)
    printf("%s ", argv[i]);
  return argc - 2;
}
```

Qu'est-ce qui se passera si on lance la commande suivante dans un environnement Linux?

```
$ gcc -c prog.c
```

- ☐ Le programme s'exécutera normalement
- ☐ Un fichier nommé `a.out` sera produit
- ■ Un fichier nommé `prog.o` sera produit
- ☐ Un fichier nommé `prog` sera produit

Q2. /3 (Suite de la question précédente) En reprenant le programme de la question précédente, qu'est-ce qui sera affiché sur la sortie standard si on lance les commandes suivantes?

```
$ gcc prog.c -o prog
$ ./prog -o a b c 4
```

Réponse: `...../prog -o a b c 4 .....`

Q3. /3 (Suite de la question précédente) En reprenant le programme de la question précédente, qu'est-ce qui sera affiché sur la sortie standard si on lance les deux commandes suivantes:

```
$ gcc prog.c -o prog
$ ./prog "b c"
$ echo $?
```

- ☐ `?`
- ☐ `$?`
- ■ `0`
- ☐ `1`

Q4. /3 Quelle commande Git permet de reformuler le message du *commit* le plus récent sur la branche courante?

- ■ `git commit --amend`
- ☐ `git commit -r HEAD~1`
- ☐ `git reset --hard`
- ☐ `git reset --soft`

Q5. /3 Que désigne l'identifiant `HEAD` pour le logiciel Git?

- ☐ La branche principale d'un dépôt
- ■ La référence locale courante
- ☐ Le dépôt principal du projet
- ☐ Le plus récent *commit* de la branche `master`

Q6. /3 (Suite de la question précédente) Vous vous trouvez actuellement sur la branche `master`. Il existe une autre branche, nommée `test`, que vous souhaitez supprimer, car vous n'en avez plus besoin. Quelle commande devriez-vous utiliser?

- ☐ `git checkout test --delete`
- ■ `git branch -D test`
- ☐ `git remove branch test`
- ☐ `git switch test`

Q7. /4 Vous vous trouvez dans un répertoire qui ne contient que le fichier `Makefile` suivant:

```
a:
  echo a
  rm -f c

b: a c
  cp d c
  cp c b

c: a
  touch d

d: b c
  echo d
```

Vous entrez la commande suivante:

```
$ make d
```

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

Après avoir exécuté `make d`, le répertoire contiendra...

(a) ...un fichier nommé `a` V / (F)

(b) ...un fichier nommé `b` (V) / F

(c) ...un fichier nommé `c` (V) / F

(d) ...un fichier nommé `d` (V) / F

Q8. /4 Sachant que la valeur ASCII du caractère `'A'` est `65`, qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>

int main(void) {
  printf("%d\n%d\n%d\n%c\n",
         0x10, 010,
         'D' - 1, 'A' + 2);
  return 0;
}
```

(a) Ligne 1: .....16.....

(b) Ligne 2: .....8.....

(c) Ligne 3: .....67.....

(d) Ligne 4: .....C.....

Q9. /4 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
int main(void) {
  unsigned char x = 'A';
  char y = 'B', z = 255;
  printf("%d\n%u\n%d\n%c\n",
         x, y, (signed char)z, x + 1);
  return 0;
}
```

(a) Ligne 1: .....65.....

(b) Ligne 2: .....66.....

(c) Ligne 3: .....-1.....

(d) Ligne 4: .....B.....

Q10. /6 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
#include <string.h>
int main(void) {
  char s[10] = "inf";
  printf("%c\n%s\n", *s, s);
  strcat(s, "3135");
  printf("%c\n%d\n%s\n",
         s[3], s[8], s + 3);
  strcpy(s + 2, "m");
  printf("%s\n", s);
  return 0;
}
```

(a) Ligne 1: .....i.....

(b) Ligne 2: .....inf.....

(c) Ligne 3: .....3.....

(d) Ligne 4: .....Une valeur indéterminée.....

(e) Ligne 5: .....3135.....

(f) Ligne 6: .....inm.....

Q11. /6 Considérez la fonction suivante:

```
bool f(const char* s, const char* t) {
  while (*s != '\0' && *t != '\0') {
    if (*s != *t)
      return false;
    ++s;
    ++t;
  }
  return *s == '\0';
}
```

Pour chacune des paires de valeurs `s` et `t` suivantes, quelle valeur sera retournée par la fonction `f`? Répondez en remplissant le tableau ci-bas:

s	t	f(s, t)
""	""	true
""	"a"	true
"a"	""	false
"ab"	"cb"	false
"abc"	"ab"	false
"ab"	"abc"	true

Q12. /2 (Suite de la question précédente) Quel nom devrait-on donner à la fonction `f` pour améliorer la lisibilité du code?

Réponse: .....`est_prefixe`.....

Q13. /4 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>

union U {
  double d;
  char* e[2];
};

struct S {
  double* b;
  union U* c[2];
};

int main(void) {
  int x, *y;
  printf("%ld\n%ld\n%ld\n%ld\n",
         sizeof(x), sizeof(y),
         sizeof(struct S),
         sizeof(union U));
  return 0;
}
```

(a) Ligne 1: .....`4`.....

(b) Ligne 2: .....`8`.....

(c) Ligne 3: .....`24`.....

(d) Ligne 4: .....`16`.....

Q14. /2 Considérez le rapport de tests suivants exécuté par Bats:

```
1..4
ok 1 - Ouverture du fichier
not ok 2 - Première ligne du fichier
ok 3 - Reste du fichier
not ok 4 - Résumé # TODO
```

Comment appelle-t-on la première ligne de ce rapport?

- ☐ Intervalle
- ☐ Résumé
- ■ Plan
- ☐ Suite

Q15. /2 (Suite de la question précédente) Quel est le protocole auquel se conforme le rapport précédent?

- ☐ BAP
- ■ TAP
- ☐ BAT
- ☐ TDD

Q16. /3 Considérez le test Bats suivant:

```
@test "Un test Bats" {
  run echo 'magie!'
  # TODO
}
```

Par quoi doit-on remplacer `# TODO` pour que le test échoue?

- ■ `assert_line --partial 'aie'`
- ☐ `assert_output --partial 'magie!'`
- ☐ `assert_output --regexp 'm.*[ag]'`
- ☐ `assert_success`

Q17. /3 Considérez le test Bats suivant:

```
@test "Un test Bats" {
  # TODO
  assert_line --regexp '[ab].*c'
}
```

Par quoi doit-on remplacer `# TODO` pour que le test réussisse?

- ☐ `echo ab`
- ☐ `echo c a`
- ■ `echo bac`
- ☐ `echo cab`

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

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

int compare(const void* s1,
            const void* s2) {
  const char **s1p = (const char**)s1,
             **s2p = (const char**)s2;
  unsigned int n1 = strlen(*s1p),
               n2 = strlen(*s2p);
  if (n1 != n2)
    return n1 - n2;
  return strcmp(*s1p, *s2p);
}

int main(void) {
  char* t[4] = {"b", "ab", "a", "aab"};
  qsort(t, 4, sizeof(char*), compare);
  for (int i = 0; i < 4; ++i)
    printf("%s ", t[i]);
  printf("\n");
  return 0;
}
```

Réponse: ..... `a b ab aab` .....

Q19. /4 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
int main(void) {
  int a[] = {4, 3, 2, 1};
  int* b[] = {a, a + 3, a + 1};
  printf("%d\n%d\n",
         **b, *(*(b + 1) - 2));
  ++(*b);
  **b += 3;
  printf("%d\n%d\n", **b, **(b + 1));
  return 0;
}
```

(a) Ligne 1: .....`4`.....

(b) Ligne 2: .....`3`.....

(c) Ligne 3: .....`6`.....

(d) Ligne 4: .....`1`.....

Q20. /4 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>

int plus_1(int x) {
  return x + 1;
}

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

int it(int (*f)(int),
       int x,
       unsigned int n) {
  for (unsigned int i = 0; i < n; ++i)
    x = f(x);
  return x;
}

int main(void) {
  printf("%d\n", it(plus_1, 0, 5));
  printf("%d\n", it(plus_1, 2, 100));
  printf("%d\n", it(carre, 1, 8));
  printf("%d\n", it(carre, 2, 2));
  return 0;
}
```

(a) Ligne 1: .....`5`.....

(b) Ligne 2: .....`102`.....

(c) Ligne 3: .....`1`.....

(d) Ligne 4: .....`16`.....

Q21. /10 Considérez le programme incomplet suivant:

```
#include <stdio.h>

/**
 * Itère sur les caractères 'a' à 'e' de façon cyclique selon un décalage donné
 *
 * Si le décalage donné est < 0, alors l'itérateur est réinitialisé et retourne
 * le premier caractère ('a').
 *
 * @param i  Le décalage à utiliser
 * @return   Le prochain caractère
 */
char char_cycle_abcde(int i) {
  // TODO
}

int main(void) {
  int pas[] = {0, 1, 2};
  for (int p = 0; p < 3; ++p) {
    printf("%c ", char_cycle_abcde(-1));
    for (int i = 0; i < 9; ++i)
      printf("%c ", char_cycle_abcde(pas[p]));
    printf("\n");
  }
}
```

Complétez la fonction `char_cycle_abcde` en respectant le comportement décrit dans sa *docstring*. On s'attend en particulier à ce qu'à l'exécution du programme, on affiche ce qui suit sur la sortie standard:

```
a a a a a a a a a a
a b c d e a b c d e
a c e b d a c e b d
```

Suggestion: Utilisez le mot réservé `static`.
Note: La lisibilité et l'efficacité de votre solution seront prises en considération dans l'évaluation.

Réponse: Il suffit d'utiliser une variable statique `j`, initialisée à `0`, pour mémoriser l'indice auquel on est rendu dans le cycle. Lorsque `i < 0`, on s'assure de réinitialiser l'indice `j`. Dans le cas contraire, on met à jour l'indice, en faisant attention d'appliquer l'opérateur modulo pour ramener la valeur entre 0 et 4. Puis on retourne le caractère correspondant.

```
char char_cycle_abcde(int i) {
  static int j = 0;
  if (i < 0)
    j = 0;
  else
    j = (j + i) % 5;
  return 'a' + j;
}
```

Q22. /20 Considérez le programme incomplet suivant:

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

// La longueur maximum d'un mot
#define LONGUEUR_MAX_MOT 30
// La longueur maximum d'une définition
#define LONGUEUR_MAX_DEFINITION 200
// Le nombre de définitions maximum d'un dictionnaire
#define NB_DEFINITIONS_MAX 100

// Types
// -----

// Un dictionnaire
struct Dictionnaire {
  // Les mots définis
  char mots[NB_DEFINITIONS_MAX][LONGUEUR_MAX_MOT];
  // Les définitions
  char definitions[NB_DEFINITIONS_MAX][LONGUEUR_MAX_DEFINITION];
  // Le nombre de définitions
  unsigned int nb_definitions;
};

// Fonctions
// ---------

/**
 * Initialise un dictionnaire vide
 *
 * @param d  Le dictionnaire à initialiser
 */
void dictionnaire_initialiser(struct Dictionnaire* d) {
  d->nb_definitions = 0;
}

/**
 * Affiche un dictionnaire sur la sortie standard
 *
 * Note: les mots sont affichés en ordre lexicographique.
 *
 * @param d  Le dictionnaire à afficher
 */
void dictionnaire_afficher(const struct Dictionnaire* d) {
  // TODO 1
}
```

```c
/**
 * Indique si un dictionnaire contient un mot donné
 *
 * @param d    Le dictionnaire
 * @param mot  Le mot
 * @return     true si et seulement si le dictionnaire contient le mot
 */
int dictionnaire_contient_mot(const struct Dictionnaire* d,
                              const char* mot) {
  // TODO 2
}

/**
 * Ajoute une entrée (un mot et sa définition) dans un dictionnaire
 *
 * Note: si le mot défini donné existe déjà dans le dictionnaire, la définition
 * existante est écrasée par la nouvelle.
 *
 * @param d           Le dictionnaire dans lequel on insère l'entrée
 * @param mot         Le mot qu'on définit
 * @param definition  La définition du mot
 */
void dictionnaire_ajouter_entree(struct Dictionnaire* d,
                                 const char* mot,
                                 const char* definition) {
  // TODO 3
}

// Main
// ----

int main(void) {
  struct Dictionnaire d;
  dictionnaire_initialiser(&d);
  dictionnaire_afficher(&d);
  dictionnaire_ajouter_entree(&d, "pomme", "un fruit rouge");
  dictionnaire_afficher(&d);
  dictionnaire_ajouter_entree(&d, "banane", "un fruit jaune");
  dictionnaire_afficher(&d);
  dictionnaire_ajouter_entree(&d, "banane", "un long fruit jaune");
  dictionnaire_afficher(&d);
  dictionnaire_ajouter_entree(&d, "melon", "un fruit rond");
  dictionnaire_afficher(&d);
  const char* mots[] = {"banane", "fraise", "pomme", "melon"};
  for (unsigned int i = 0; i < 3; ++i)
    printf("Est-ce que le dictionnaire contient le mot \"%s\"? %s\n",
           mots[i],
           dictionnaire_contient_mot(&d, mots[i]) ? "oui" : "non");
  return 0;
}
```

Complétez les fonctions

- `dictionnaire_afficher` (`// TODO 1`),
- `dictionnaire_contient_mot` (`// TODO 2`) et
- `dictionnaire_ajouter_entree` (`// TODO 3`)

de telle sorte qu'elles respectent le comportement décrit dans les commentaires. En particulier, on s'attend à ce que le programme affiche ceci sur la sortie standard lors de l'exécution:

```
Un dictionnaire de 0 définition
Un dictionnaire de 1 définition
  pomme: un fruit rouge
Un dictionnaire de 2 définitions
  banane: un fruit jaune
  pomme: un fruit rouge
Un dictionnaire de 2 définitions
  banane: un long fruit jaune
  pomme: un fruit rouge
Un dictionnaire de 3 définitions
  banane: un long fruit jaune
  melon: un fruit rond
  pomme: un fruit rouge
Est-ce que le dictionnaire contient le mot "banane"? oui
Est-ce que le dictionnaire contient le mot "fraise"? non
Est-ce que le dictionnaire contient le mot "pomme"? oui
```

Suggestion: N'hésitez pas à introduire des fonctions supplémentaires si nécessaire. *Note:* La lisibilité et l'efficacité de votre solution seront prises en considération dans l'évaluation.

Réponse: Tout d'abord, on note qu'il est pratique d'introduire une fonction `dictionnaire_indice_mot` qui retourne l'indice auquel se trouve un mot donné ou la valeur spéciale `-1` dans le cas où le mot ne se trouve pas dans le dictionnaire:

```
/**
 * Retourne l'indice d'un mot dans un dictionnaire
 *
 * Note: si le mot ne se trouve pas dans le dictionnaire, la valeur -1 est
 * retournée.
 *
 * @param d    Le dictionnaire
 * @param mot  Le mot
 * @return     L'indice auquel se trouve le mot ou -1
 */
int dictionnaire_indice_mot(const struct Dictionnaire* d,
                            const char* mot) {
  for (int i = 0; i < d->nb_definitions; ++i)
    if (strcmp(d->mots[i], mot) == 0)
      return i;
  return -1;
}
```

Ensuite, il est relativement direct d'implémenter chacune des trois fonctions:

```
// TODO 1:

void dictionnaire_afficher(const struct Dictionnaire* d) {
  printf("Un dictionnaire de %d définition%s\n",
         d->nb_definitions,
         d->nb_definitions > 1 ? "s" : "");
  for (unsigned int i = 0; i < d->nb_definitions; ++i)
    printf("  %s: %s\n", d->mots[i], d->definitions[i]);
}

// TODO 2:

int dictionnaire_contient_mot(const struct Dictionnaire* d,
                              const char* mot) {
  return dictionnaire_indice_mot(d, mot) != -1;
}

// TODO 3:

void dictionnaire_ajouter_entree(struct Dictionnaire* d,
                                 const char* mot,
                                 const char* definition) {
  int i = dictionnaire_indice_mot(d, mot);
  if (i != -1) {
    strcpy(d->definitions[i], definition);
  } else {
    i = d->nb_definitions;
    while (i > 0 && strcmp(d->mots[i - 1], mot) > 0) {
      strcpy(d->mots[i], d->mots[i - 1]);
      strcpy(d->definitions[i], d->definitions[i - 1]);
      --i;
    }
    strcpy(d->mots[i], mot);
    strcpy(d->definitions[i], definition);
    ++d->nb_definitions;
  }
}
```