Total /50

Identification

Prénom: .....

Nom: .....

Code permanent: .....

Instructions

- L'examen dure 3 heures
- Vous avez droit à une feuille de notes recto-verso, au format lettre ($8\ 1/2 \times 11$)
- 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, en utilisant une croix ($\times$) ou un crochet ($\checkmark$)
- Pour chacune des questions, vous pouvez supposer que:
  - `sizeof(int)` retourne 4;
  - `sizeof(char)` retourne 1;
  - `sizeof(bool)` retourne 1 et
  - `sizeof(double)` retourne 8 et
  - `sizeof(T*)` retourne 8, peu importe `T`.

Q1. /1 Considérez le programme C suivant:

```
#include <stdio.h>
int main(void) {
  printf("prog\n");
  return 1;
}
```

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

```
$ gcc prog.c
```

- ☐ La ligne `prog` sera affichée
- ☐ Rien du tout
- ☐ Une erreur de compilation sera affichée
- ■ Un exécutable nommé `a.out` sera produit

Q2. /1 (Suite de la question précédente) En reprenant le programme de la question précédente, quelle commande permet de produire un exécutable nommé `prog`?

- ☐ `gcc prog`
- ☐ `gcc prog.c`
- ■ `gcc prog.c -o prog`
- ☐ `gcc prog.c prog`

Q3. /1 (Suite de la question précédente) En supposant que l'exécutable `prog` a été produit correctement, qu'est-ce qui sera affiché sur la sortie standard par la deuxième commande si on exécute les deux commandes suivantes:

```
$ ./prog
$ echo $?
```

☐ `$?` ☐ `0` ■ `1` ☐ `?`

Q4. /1 Vous travaillez sur un projet en local, versionné avec Git. Vous avez apporté différentes modifications à plusieurs fichiers, mais vous avez oublié de les sauvegarder (*commit*) avec Git. Vous souhaitez partager seulement quelques-unes des modifications apportées au fichier `README.md` aux autres membres de votre équipe. Quelle est la première commande que vous devriez entrer?

- ■ `git add -p`
- ☐ `git add README.md`
- ☐ `git commit -a`
- ☐ `git commit README.md`

Q5. /1 Vous travaillez sur un projet en local, versionné avec Git. Des membres de votre équipe ont apporté des changements sur la branche `master` du dépôt distant partagé nommé `origin`. Quelle commande vous permet de télécharger les modifications apportées par les membres de votre équipe?

- ☐ `git download master`
- ☐ `git download origin`
- ☐ `git fetch master`
- ■ `git fetch origin`

Q6. /1 (Suite de la question précédente) Vous avez réussi à télécharger les modifications les plus récentes du dépôt distant. Vous vous trouvez sur la branche `corrige-bogue` et l'état actuel de votre projet est propre (*clean*). Vous souhaitez maintenant synchroniser votre branche locale `master` avec la branche distante `master`. Quelle est la première commande que vous devriez entrer?

- ■ `git checkout master`
- ☐ `git merge master`
- ☐ `git pull master`
- ☐ `git sync master`

Q7. /3 Vous vous trouvez dans un répertoire qui ne contient que le fichier `Makefile` suivant:

```
.PHONY: c cl

c: b2
  cat b2

b1:
  echo bonjour > b1

b2: b1
  cat b1 > b2

cl:
  rm -f b1 b2
```

Vous entrez la commande suivante:

```
$ make
```

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

(a) Un fichier nommé `b1` sera produit (V) / F

(b) Un fichier nommé `b2` sera produit (V) / F

(c) Un fichier nommé `c` sera produit V / (F)

Q8. /3 Quelles sont les 3 valeurs qui seront affichées sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
int main(void) {
  int i = 012, j = 210, k = 0x12;
  printf("%d %d %d\n", i, j, k);
  return 0;
}
```

(a) Valeur 1: ..... **10** .....

(b) Valeur 2: ..... **210** .....

(c) Valeur 3: ..... **18** .....

Q9. /3 Quelles sont les 3 valeurs qui seront affichées sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
int main(void) {
  unsigned char i, j, k;
  i = 128; j = 2 * i; k = j - 1;
  printf("%d %d %d\n", i, j, k);
  return 0;
}
```

(a) Valeur 1: ..... **128** .....

(b) Valeur 2: ..... **0** .....

(c) Valeur 3: ..... **255** .....

Q10. /3 Quelles sont les 3 valeurs qui seront affichées sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
#include <string.h>
int main(void) {
  char s[] = "abracadabra";
  s[4] = '\0'; printf("%s ", s);
  strcat(s, "sif"); printf("%s ", s);
  char t[] = "magie";
  strncpy(t, s, 4); printf("%s ", t);
  return 0;
}
```

(a) Valeur 1: ..... **abra** .....

(b) Valeur 2: ..... **abrasif** .....

(c) Valeur 3: ..... **abrae** .....

Q11. /1 Quel nom devrait-on donner à la fonction `f` pour améliorer la lisibilité du code?

```
#include <stdio.h>
void f(int* i) {
  *i *= 3;
}
int main(void) {
  int i = 2;
  f(&i);
  return 0;
}
```

- ☐ `multiple3`
- ☐ `multiplier3`
- ☐ `triple`
- ■ `tripler`

Q12. /2 Considérez le programme C suivant:

```
#include <stdio.h>
#include <string.h>
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; ++i) {
    size_t n = strlen(argv[i]);
    printf("%c", argv[i][n - 1]);
  }
  return 0;
}
```

Supposons qu'on le compile en un exécutable nommé `main`. Qu'est-ce qui sera affiché sur la sortie standard si on entre la commande suivante?

```
$ ./main un deux "trois quatre"
```

Réponse: ..... **nnxe** .....

Q13. /2 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  char* s;
  int i = strtol("1ab2", &s, 10);
  printf("%d %s\n", i, s);
  return 0;
}
```

Réponse: ..... **1 ab2** .....

Q14. /2 Quelles sont les 4 valeurs qui seront affichées sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
typedef struct {
  int a;
  double* b;
  char c[8];
} S;
typedef union {
  struct S* d[2];
  int* e;
} U;
int main(void) {
  U* u[2];
  printf("%ld %ld %ld %ld\n",
         sizeof(char*), sizeof(S),
         sizeof(U), sizeof(u));
  return 0;
}
```

(a) Valeur 1: ..... **8** .....

(b) Valeur 2: ..... **20** (ou **24**) .....

(c) Valeur 3: ..... **16** .....

(d) Valeur 4: ..... **16** .....

Q15. /2 Quelles sont les 4 valeurs qui seront affichées sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
#include <string.h>
int main(void) {
  char s[] = "ab,c,.de.";
  char* t = strtok(s, ",.");
  while (t != NULL) {
    printf("%s ", t);
    t = strtok(NULL, ",.");
  }
  printf("%s\n", s);
}
```

(a) Valeur 1: ..... **ab** .....

(b) Valeur 2: ..... **c** .....

(c) Valeur 3: ..... **de** .....

(d) Valeur 4: ..... **ab** .....

Q16. /1 Quel nom donne-t-on au commentaire qui apparaît dans le code source au-dessus d'une fonction pour documenter son fonctionnement?

- ■ *docstring*
- ☐ Javadoc
- ☐ Markdown
- ☐ *readme*

Q17. /2 Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
#include <stdlib.h>
struct P {
  int x;
  int y;
};
int compare(const void* p1,
            const void* p2) {
  const struct P
    *p1c = (const struct P*)p1,
    *p2c = (const struct P*)p2;
  if (p1c->y != p2c->y)
    return p1c->y - p2c->y;
  return p1c->x - p2c->x;
}
int main(void) {
  struct P ps[] = {{1, 0}, {0, 1},
                   {1, 1}, {0, 0}};
  qsort(ps, 4, sizeof(struct P),
        compare);
  for (int i = 0; i < 4; ++i)
    printf("(%d,%d) ",
           ps[i].x, ps[i].y);
  printf("\n");
  return 0;
}
```

Réponse: ..... **(0,0) (1,0) (0,1) (1,1)** .....

Q18. /2 Quelles sont les 4 valeurs qui seront affichées 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, *c = a + 5;
  printf("%d %d ", *b, *(c - 2));
  b += 3;
  printf("%d %d\n", *b, b < c ? 4 : 3);
  return 0;
}
```

(a) Valeur 1: ..... **4** .....

(b) Valeur 2: ..... **1** .....

(c) Valeur 3: ..... **1** .....

(d) Valeur 4: ..... **4** .....

Q19. /2 Quelles sont les 2 valeurs qui seront affichées sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>
#include <stdbool.h>
int f() {
  static bool b;
  static int i;
  b = !b;
  ++i;
  return b ? i : i + 1;
}
int main(void) {
  printf("%d %d\n", f(), f());
  return 0;
}
```

(a) Valeur 1: ..... **1 ou 3** .....

(b) Valeur 2: ..... **3 ou 1** .....

Q20. /1 Parmi les stratégies suivantes, laquelle contribue à prévenir les bogues?

- ■ Développer de façon incrémentale
- ☐ Raisonner en marche-arrière
- ☐ Regrouper les bogues similaires
- ☐ Utiliser un débogueur

Q21. /5 Donnez l'implémentation d'une fonction en C dont la signature est

```
bool est_heure_valide(const char* s)
```

qui indique si une chaîne de caractères donnée représente une heure de la journée valide au format `HH:MM:SS`, qui doit être entre `00:00:00` et `23:59:59`. Votre fonction doit être correcte peu importe la chaîne de caractères `s`, mais vous pouvez prendre pour acquis que `s` est bien formée (c'est-à-dire qu'elle termine par le caractère `'0'`). Indiquez quelles inclusions (`#include`) vous devez ajouter pour implémenter votre fonction. *Note:* La lisibilité et l'efficacité de votre solution seront prises en considération dans l'évaluation.

Une solution possible est:

```
#include <stdbool.h>
#include <ctype.h>

bool est_entre(char c, char c1, char c2) {
  return c >= c1 && c <= c2;
}

bool est_heure_valide(const char* s) {
  return s != NULL &&
         est_entre(s[0], '0', '2') &&
         est_entre(s[1], '0', '9') &&
         (s[0] != '2' || est_entre(s[1], '0', '3')) &&
         s[2] == ':' &&
         est_entre(s[3], '0', '5') &&
         est_entre(s[4], '0', '9') &&
         s[5] == ':' &&
         est_entre(s[6], '0', '5') &&
         est_entre(s[7], '0', '9') &&
         s[8] == '\0';
}
```

Q22. /3 Proposez un cadre de tests pour la fonction

```
bool est_heure_valide(const char* s)
```

définie à la question précédente. Plus précisément, donnez une liste de valeurs possibles pour `s` qui vous permettent de maximiser la couverture et minimiser la redondance de votre cadre de tests. Justifiez brièvement ce que chaque test vous permet de vérifier et en quoi la redondance est minimisée.

Les chaînes suivantes devraient être reconnues comme des heures valides:

- `"00:00:00"`: cas limite inférieur
- `"09:59:59"`: cas limite avant 10:00:00
- `"19:59:59"`: cas limite avant 20:00:00
- `"23:59:59"`: cas limite supérieur

Les chaînes qui suivent devraient être reconnues comme non valides:

- `""`: chaîne trop courte (chaîne vide)
- `"00:00:00:"`: chaîne trop longue
- `"24:00:00"`: heures trop grandes
- `"00:60:00"`: minutes trop grandes
- `"00:00:60"`: secondes trop grandes
- `"a0:00:00"`: 1er caractère invalide
- `"0a:00:00"`: 2e caractère invalide
- `"00!00:00"`: 3e caractère invalide
- `"00:a0:00"`: 4e caractère invalide
- `"00:0a:00"`: 5e caractère invalide
- `"00:00!00"`: 6e caractère invalide
- `"00:00:a0"`: 7e caractère invalide
- `"00:00:0a"`: 8e caractère invalide

De cette facon, on couvre plusieurs cas limites, on s'assure de vérifier tous les branchements dans la condition logique qui apparaît dans la solution précédente. Chaque test vérifie une possibilité différente, donc il n'y a pas vraiment de redondance.

Q23. /7 Considérez le programme incomplet suivant, qui permet de manipuler et d'afficher des expressions numériques:

```
#include <stdio.h>

// Un type d'expression
enum Type {
  // Nombre
  NOMBRE,
  // Opération
  OPERATION
};

// Un opérateur
enum Operateur {
  // Multiplication
  MULT,
  // Addition
  PLUS,
};

// Une expression
struct Expression {
  // Type
  enum Type type;
  union {
    // Un nombre
    int nombre;
    // Une sous-expression
    struct {
      enum Operateur operateur;
      const struct Expression* gauche;
      const struct Expression* droite;
    } operation;
  };
};

/**
 * Retourne le caractère représentant l'opérateur
 *
 * @param operateur  L'opérateur
 */
char operateur(enum Operateur operateur) {
  switch (operateur) {
    case MULT: return '*';
    case PLUS: return '+';
  };
  return '?';
}

/**
 * Initialise une expression en tant que nombre
 *
 * @param expression  L'expression à initialiser
 * @param nombre      Le nombre à utiliser
```

```c
 */
void initialiser_nombre(struct Expression* expression, int nombre) {
  expression->type = NOMBRE;
  expression->nombre = nombre;
}

/**
 * Initialise une expression en tant qu'opération
 *
 * @param expression  L'expression à initialiser
 * @param operateur   L'opérateur à utiliser
 * @param gauche      La sous-expression à gauche
 * @param droite      La sous-expression à droite
 */
void initialiser_operation(struct Expression* expression,
                           enum Operateur operateur,
                           const struct Expression* gauche,
                           const struct Expression* droite) {
  expression->type = OPERATION;
  expression->operation.operateur = operateur;
  expression->operation.gauche = gauche;
  expression->operation.droite = droite;
}

/**
 * Affiche une expression sur la sortie standard
 *
 * @param expression  L'expression à afficher
 */
void afficher_expression(const struct Expression* expression) {
  // TODO 1
}

/**
 * Retourne la valeur d'une expression
 *
 * @param expression  L'expression à évaluer
 * @return            La valeur de l'expression
 */
int valeur_expression(const struct Expression* expression) {
  // TODO 2
}

int main(void) {
  struct Expression e1, e2, e3, e4;
  initialiser_nombre(&e1, 1);
  initialiser_nombre(&e2, 2);
  initialiser_operation(&e3, PLUS, &e1, &e2);
  initialiser_operation(&e4, MULT, &e3, &e2);
  afficher_expression(&e1); printf(" = %d\n", valeur_expression(&e1));
  afficher_expression(&e2); printf(" = %d\n", valeur_expression(&e2));
  afficher_expression(&e3); printf(" = %d\n", valeur_expression(&e3));
  afficher_expression(&e4); printf(" = %d\n", valeur_expression(&e4));
  return 0;
}
```

Complétez les fonctions `afficher_expression` (`// TODO 1`) et `valeur_expression` (`// TODO 2`) 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:

```
1 = 1
2 = 2
(1 + 2) = 3
((1 + 2) * 2) = 6
```

Note: La lisibilité et l'efficacité de votre solution seront prises en considération dans l'évaluation.

Voici une solution possible pour chacune des deux fonctions:

```
void afficher_expression(const struct Expression* expression) {
  switch (expression->type) {
    case NOMBRE:
      printf("%d", expression->nombre);
      break;
    case OPERATION:
      printf("(");
      afficher_expression(expression->operation.gauche);
      printf(" %c ", operateur(expression->operation.operateur));
      afficher_expression(expression->operation.droite);
      printf(")");
      break;
  }
}

int valeur_expression(const struct Expression* expression) {
  switch (expression->type) {
    case NOMBRE:
      return expression->nombre;
    case OPERATION:
      int g = valeur_expression(expression->operation.gauche),
          d = valeur_expression(expression->operation.droite);
      switch (expression->operation.operateur) {
        case PLUS: return g + d;
        case MULT: return g * d;
      };
  }
  return 0;
}
```