Contrôle continu n^o 99 FAKE

Tout document, calculatrice, ordinateur ou moyen de communication est interdit.
Les exercices sont indépendants. Les codes demandés doivent être écrits en C.
Vous répondrez sur les feuilles placées à la fin de l'énoncé, en renseignant pour chacune d'elle vos prénom, nom et numéro d'étudiant.
Durée de l'épreuve : 24h.

Exercice 1

1) 😎 On veut écrire une fonction `sqrt_int` prenant en paramètre un entier de type int, posititif ou nul et renvoyant la partie entière de la racine carrée de cet entier.
   a) Dans quel cas on ne peut pas calculer cette valeur ?
   b) Écrire la partie commentaire que vous allez placer avant le prototype de la fonction. En particulier, donner l'assertion d'entrée et l'assertion de sortie.
   c) Si ce n'est pas le cas, reécrire l'assertion de sortie en utilisant le fait que la partie entière de la racine carrée de n est l'unique entier positif i tel que $i^2 \leqslant n$ et $(i+1)^2 > n$.
   d) Écrire le prototype de `sqrt_int`.
2) 🙄 Écrire la définition de la fonction `sqrt_int`.
   Indication : Vous pourrez utiliser une variable `i` et écrire une boucle respectant l'invariant

   ```
   IB : (i - 1) * (i - 1) < n
   ```

3) 😈 Écrire une fonction de prototype

   ```
   bool scan_pos (int*p);
   ```

   qui demande à l'utilisateur d'entrer un entier positif et renvoie celui-ci. Si l'utilisateur donne en entrée un entier strictement négatif, la fonction lui redemande jusqu'à ce que l'entier soit positif. En cas d'erreur de saisie (autre que le signe), la fonction renvoie `false`, sinon elle renvoie `true`.
4) 🙄 Donnez un corps à la fonction `main` afin de tester les deux fonctions que vous avez écrite.

Exercice 2
Le calendrier grégorien est celui que nous utilisons depuis le 15 Octobre 1582.
Les mois de Janvier, Mars, Mai, Juillet, Aout, Octobre et Décembre comptent 31 jours.
Les mois d'Avril, Juin, Septembre et Novembre comptent 30 jours.
Le mois de Février compte 29 jours si l'année et bisextile et 28 sinon.
Une année est bisextile si elle est divisible par 4 mais pas par 100 ou si elle est divisible par 400.
Par exemple : L'année 2003 n'est pas une année bisextile car elle n'est pas divisible par 4.
L'année 1900 n'est pas une année bisextile car elle est divisible par 100. L'année 2004 est bisextile car elle est divisible par 4 mais par par 100. L'année 2000 est bisextile car elle est divisible par 400.
Le but de cet exercice est d'écrire un programme permettant de tester si une date fournie par l'utilisateur sous la forme `jj/mm/aaaa` est valide dans le calendrier grégorien. Les mois sont représentés par des entiers Janvier = 1, Février = 2 etc.

1)  Une date `jj/mm/aaaa` est valide si le jour (`jj`) est un entier positif, le mois (`mm`) est un entier positif inférieur ou égal à 12, la date se situe après le 15 octobre 1582 et qu'elle respecte les règles énoncées ci-dessus.
   Décrire, à l'aide d'arbres la structure de tests permettant de décider si une date est valide. Il est conseillé d'utiliser plusieurs arbres successifs afin de simplifier la méthode.

2)  En déduire une fonction de prototype
   `bool date_valide(int jj, int mm, int aaaa);`
   renvoyant `true` si la date est valide et `false` sinon. Vous vous efforcerez de suivre l'arbre de test que vous avez décrit dans la question précédente. Vous pourrez supprimer les mots clef `else` si ils vous semblent inutiles afin de ne pas alourdir l'indentation.

3)  Donnez une version de cette fonction qui n'utilise pas d'instruction `if` ni d'instruction `switch`. On s'efforcera de préserver la lisibilité du programme.

Solutions

Solution de l'exercice 1

1) a) Lorsque l'entier est strictement négatif.

b)

```
/* Déclarations des fonctions et des macros */
/* sqrt_int:
 * Renvoie la partie entiere de la racine carrée de l'entier
 * passé en paramètre.
 * Entrée : un entier n de type int
 * Sortie : une valeur de type entière
 * A.E. : n >= 0
 * A.S. : sqrt_int == partie entière de la racine carrée de n
 */
```

c)

```
 * A.S. :  sqrt_int^2 <= n && (sqrt_int + 1) > n
```

d)

```
int sqrt_int(int n);
```

2)

```
int sqrt_int(int n) {
  assert(n >= 0);
  /* IB : (i - 1) * (i - 1) < n
   */
  int i = 1;
  while (i * i <= n ) {
    ++i;
  }
/* on peut aussi écrire
 * for (; i * i <= n; ++i) {
 * }
 */

/* ici on a (i - 1) * ( i - 1) <= n && i * i > n
 * ce qui implique que i - 1 est la partie entiere
 * de la racine carré de n
 */

  return i - 1;
}
```

3)

```
 /* scan_pos:
  * Demande à l'utilisateur d'entrer un entier positif
  * et renvoie celui-ci.
  * Si l'utilisateur donne en entrée un entier strictement
```

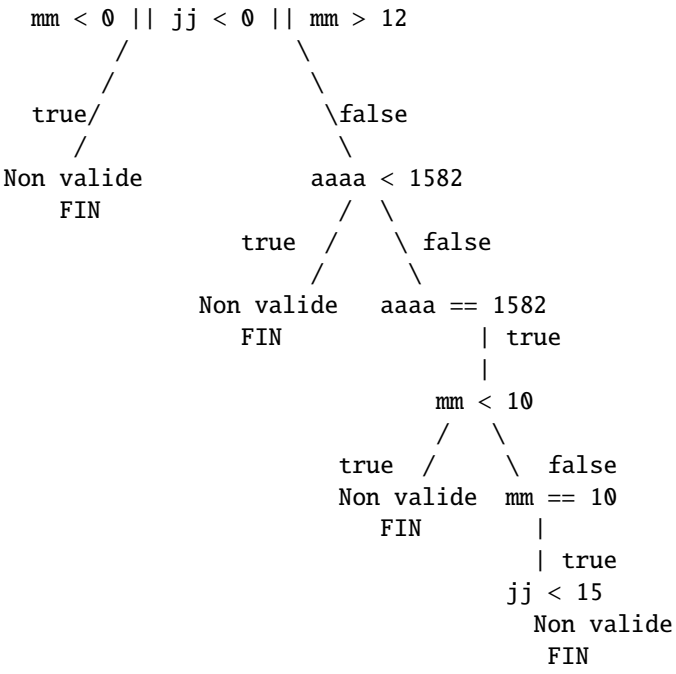
```
  * négatif, la fonction lui redemande jusqu'à ce que l'entier
  * soit positif.
  * En cas d'erreur de saisie, la fonction renvoie false,
  * sinon elle renvoie true
  * Entrée : p de type int* l'adresse d'un entier de type int
  * Sortie : une valeur de type bool
  * A.E. :  p est une adresse valide et modifiable.
  * A.S. : (scan_pos == true && *p == entier saisi par l'utilisateur && *p > 0)
  *        || (!scan_pos == il y a eu une erreur sur l'entrée standard)
  */
bool scan_pos (int*p);
....
....
bool scan_pos(int* p) {
  do {
    printf("Entrez un entier positif ou nul :\n");
    if (scanf("%d", p) != 1) {
      return false;
    }
    if (*p < 0) {
      printf(" Erreur de saisie, l'entier est négatif!\n");
    }
  } while (*p < 0);
  /* IB : *p n'a pas été saisi || *p < 0 */

  return true;
}
```

4)

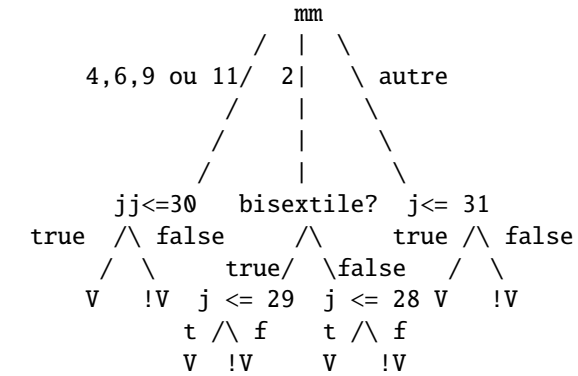
```
/* Fonction principale */
int main(void) {
  int n;
  if (!scan_pos(&n)) {
    printf("Erreur de saisie\n");
    return EXIT_FAILURE;
  }
  printf("sqrt(%d) = %d\n", n, sqrt_int(n));
  return EXIT_SUCCESS;
}
```

Solution de l'exercice 2

1) On commence par vérifier que la date est dans le bon intervalle. Ceci doonne l'arbre



Remarque certains des arbres sont unaires. Cela signifie qu'il n'y a pas de partie `else` dans le test et qu'il faut attendre la suite des instructions conditionnelles pour finir le travail.



Légende
`V == Valide`, `!V == Non Valide`, `t == true`, `t == false`.

2)

```
/* date_valide : prend en paramètre trois entiers représentant une date sous
 * la forme JJ/MM/AAAA, qui retourne true si elle représente une date du
 * calendrier grégorien (après le 15 octobre 1582) et false sinon
 * Entrées : jj, mm et aaaa trois entiers de type int
 * Sortie : un valeur booleenne
 * A.E. : true
 * A.S. : date_valide == la date jj/mm/aaaa est valide
 */
bool date_valide(int jj, int mm, int aaaa);
```

Puis dans la partie définition :

```

```

```
bool date_valide(int jj, int mm, int aaaa) {
/* on traite le cas des numéros qui ne peuvent pas représenter des mois
 * et des numéros de jours négatifs */
  if (mm < 0 || jj < 0 || mm > 12) {
    return false;
  }
/* on vérifie que l'on ne se situe pas avant le début du calendrier grégorien*/
  if ( aaaa < 1582) {
    return false;
  }
  if (aaaa == 1582) {
    if (mm < 10) {
      return false;
    }
    if (mm == 10) {
      if ( jj < 15) {
        return false;
      }
    }
  }
  switch (mm) {
  case 4:
  case 6:
  case 9:
  case 11:
    return jj <= 30;
  break;
  case 2:
  /* on vérifie si l'année est bissextile*/
    if ((aaaa % 4 == 0 && aaaa % 100 != 0) || (aaaa % 400 == 0)) {
      return jj <= 29;
    } else {
        return jj <= 28;
    }
  break;
  default:
    return jj <= 31;
  }
  return true;
}
```

3)

```
bool date_valide_alt(int jj, int mm, int aaaa) {
/* on traite le cas des numéros qui ne peuvent pas représenter des mois
 * et des numéros de jours négatifs */
  bool invalide = (mm < 0 || jj < 0 || mm > 12);

/* on vérifie que l'on ne se situe pas avant le début du calendrier grégorien*/
  invalide = invalide || ( aaaa < 1582);
  invalide = invalide || ( aaaa == 1582 && (mm <= 10 || (mm == 10 && jj < 15)));

/* on teste la validité du numéro du jour */
  bool mois_en_30 = (mm == 4) || (mm == 6) || (mm == 9) || (mm == 11);
```

```
  bool mois_en_31 = !mois_en_30 && mm != 2;
  bool annee_bi = (aaaa % 4 == 0 && aaaa % 100 != 0) || aaaa % 400 == 0;
  bool invalide_fevrier = (annee_bi && jj > 29) || (!annee_bi && jj > 28);

  invalide = invalide || ( mois_en_30 && jj > 30);
  invalide = invalide || (mois_en_31 && jj > 31);
  invalide = invalide || (mm == 2 && invalide_fevrier);

  return !invalide;
}
```