Solution exercice 4.8

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

char* concat(const char* prenom, const char* nom);

void clear_stdin(void);

int main(void) {

   #define TAILLE_MAX_PRENOM 20
   #define TAILLE_MAX_NOM 30
   char prenom[TAILLE_MAX_PRENOM + 1];
   char nom[TAILLE_MAX_NOM + 1];

   #define TAILLE_MAX_CHAINE_CONTROLE 15
   char chaineControlePrenom[TAILLE_MAX_CHAINE_CONTROLE + 1];
   char chaineControleNom[TAILLE_MAX_CHAINE_CONTROLE + 1];

   const char* const MOTIF = " %%%d[^\n]";
   sprintf(chaineControlePrenom, MOTIF, TAILLE_MAX_PRENOM);
   sprintf(chaineControleNom, MOTIF, TAILLE_MAX_NOM);

   printf("Entrer le prenom (max %u caract.) : ", TAILLE_MAX_PRENOM);
   scanf(chaineControlePrenom, prenom);
   clear_stdin();

   printf("Entrer le nom (max %u caract.) : ", TAILLE_MAX_NOM);
   scanf(chaineControleNom, nom);
   clear_stdin();

   char* prenom_nom = concat(prenom, nom);
   if (prenom_nom) {
      printf("La chaine \"%s\" comporte %u caracteres.\n",
             prenom_nom, (unsigned)strlen(prenom_nom));
      free(prenom_nom);
   }

   return EXIT_SUCCESS;
}

char* concat(const char* prenom, const char* nom) {
   // Construire un tableau dynamique de la taille nécessaire au
   // stockage du prénom, d'un espace, du nom et du caractère '\0'
   char* resultat = (char*) calloc(strlen(prenom) + strlen(nom) + 2, sizeof(char));
   if (resultat) {
      strcpy(resultat, prenom);
      strcat(resultat, " ");
      strcat(resultat, nom);
   }
   return resultat;
}

void clear_stdin(void) {
   int c;
   do {
      c = getchar();
   } while (c != '\n' && c != EOF);
}

// Entrer le prenom (max 20 caract.) :    Jean
// Entrer le nom (max 30 caract.) :   de la Fontaine
// La chaine "Jean de la Fontaine" comporte 19 caracteres.
```

Autre variante utilisant *fgets* :

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

void saisie(char* chaine, size_t taille);
char* concat(const char* prenom, const char* nom);
void clear_stdin(void);

int main(void) {

   #define TAILLE_MAX_PRENOM 20
   #define TAILLE_MAX_NOM 30
   char prenom[TAILLE_MAX_PRENOM + 1];
   char nom[TAILLE_MAX_NOM + 1];

   printf("Entrer le prenom (max %u caract.) : ", TAILLE_MAX_PRENOM);
   saisie(prenom, TAILLE_MAX_PRENOM);

   printf("Entrer le nom (max %u caract.) : ", TAILLE_MAX_NOM);
   saisie(nom, TAILLE_MAX_NOM);

   char* prenom_nom = concat(prenom, nom);
   if (prenom_nom) {
      printf("La chaine \"%s\" comporte %u caracteres.\n",
             prenom_nom, (unsigned)strlen(prenom_nom));
      free(prenom_nom);
   }

   return EXIT_SUCCESS;
}

void saisie(char* chaine, size_t taille) {
   fgets(chaine, (int)taille + 1, stdin);
   clear_stdin();
   // Remplace la marque de fin de ligne ('\n') év présente dans la chaîne
   // par une marque de fin de chaîne ('\0')
   for (size_t i = 0; i < taille; ++i)
      if (chaine[i] == '\n') {
         chaine[i] = '\0';
         break;
      }
}

char* concat(const char* prenom, const char* nom) {
   // Construire un tableau dynamique de la taille nécessaire au
   // stockage du prénom, d'un espace, du nom et du caractère '\0'
   char* resultat = (char*) calloc(strlen(prenom) + strlen(nom) + 2, sizeof(char));
   if (resultat) {
      strcpy(resultat, prenom);
      strcat(resultat, " ");
      strcat(resultat, nom);
   }
   return resultat;
}

void clear_stdin(void) {
   fseek(stdin, 0, SEEK_END);
}
```

Remarque Aucune des solutions ci-dessus ne gère proprement le problème des éventuels blancs excédentaires qu'il s'agirait d'éliminer (*trim*).

Solution exercice 4.9

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

// NB Ne fait rien si s vaut NULL
void inverser_1(char* s);

// NB Renvoie s si s vaut NULL ou en cas de mémoire insuffisante
char* inverser_2(const char* s);

int main(void) {

   char s1[] = "ABCD"; // Attention : inverser_1 ne fonctionnerait pas avec :
                       // const char* s1 = "ABCD" (car "ABCD" est une chaîne cste,
                       // ... donc(!) non modifiable
   printf("s1 avant inversion = %s\n", s1);
   inverser_1(s1);
   printf("s1 apres inversion = %s\n", s1);

   const char* s2 = "ABCD";
   char* s3 = inverser_2(s2);
   printf("\ns2 avant inversion = %s\n", s2);
   printf("inverse de s2 = %s\n", s3);
   printf("s2 apres inversion = %s\n", s2);

   free(s3);

   inverser_1(NULL); // Ne doit pas "planter" le programme
   inverser_2(NULL); // Idem

   return EXIT_SUCCESS;
}

void inverser_1(char* s) {
   if (s != NULL) {
      char c, *ptr = s + strlen(s) - 1;
      while (s < ptr) {
         c = *s;
         *s++ = *ptr;
         *ptr-- = c;
      }
   }
}

char* inverser_2(const char* s) {
   if (s != NULL) {
      const size_t TAILLE = strlen(s);
      char* r = (char*) calloc(TAILLE + 1, sizeof(char));
      if (r != NULL) {
         char* ptr = r + TAILLE - 1;
         for(; *s; s++)
            *ptr-- = *s;
         return r;
      }
   }
   return (char*)s;
}

// s1 avant inversion = ABCD
// s1 apres inversion = DCBA
//
// s2 avant inversion = ABCD
// inverse de s2 = DCBA
// s2 apres inversion = ABCD
```

Avantages (+) / désavantages (-) de inverser_1:

- (+) rapide car pas d'allocation dynamique
- (-) La chaîne originale étant modifiée, il n'est pas possible de passer une chaîne constante en paramètre effectif

Avantages (+) / désavantages (-) de inverser_2 :

- (+) La chaîne originale n'étant pas modifiée, il est possible de passer une chaîne constante en paramètre effectif
- (-) Plus lente car nécessite une allocation dynamique

Chapitre 5 : Fichiers

Exercice 5.1 Lecture intégrale d'un fichier texte

On suppose disposer du fichier texte suivant :

```
1
2
3
4
5
```

Ecrire le plus proprement possible un programme C qui permette d'afficher à l'écran le contenu intégral de ce fichier.

Solution exercice 5.1

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

#define NOM_FICHIER "fichier.txt"

int main(void) {
   FILE* f = fopen(NOM_FICHIER, "r");
   if (!f) { // Si ouverture du fichier impossible
      printf("Ouverture du fichier \"%s\" impossible.\n", NOM_FICHIER);
      return EXIT_FAILURE;
   } else {
      int n;
      while ( fscanf(f, "%d", &n) != EOF )
         printf("%d\n", n);
      // fermer le fichier
      fclose(f);
      return EXIT_SUCCESS;
   }
}
```

Autres variantes possibles :

```
while ( fscanf(f, "%d", &n) == 1 )
   printf("%d\n", n);

while ( !feof(f) )
   if ( fscanf(f, "%d", &n) == 1 ) {
      printf("%d\n", n);

do {
   if ( fscanf(f, "%d", &n) == 1 )
      printf("%d\n", n);
} while ( !feof(f) );


// Attention! Les variantes ci-dessous fonctionnent si le fichier
// contient au moins un entier... mais pas s'il n'en contient aucun
while (1) {
   fscanf(f, "%d", &n);
   printf("%d\n", n);
   if (feof(f)) break; // pas des plus esthétiques
}

while ( !feof(f) ) {
   fscanf(f, "%d", &n);
   printf("%d\n", n);
}
```

Exercice 5.2 Ecriture d'un fichier binaire

Ecrire un programme C permettant de stocker dans un fichier binaire nommé *personnes.dat* des données relatives à des personnes.

On suppose qu'une personne se caractérise par les propriétés suivantes :

- son nom (20 caractères max)
- son prénom (15 caractères max)
- son âge (de type *unsigned short*)

Les données des diverses personnes doivent être saisies par l'utilisateur. On conviendra que la saisie se termine dès lors que l'utilisateur entre un nom de personne vide.

Solution exercice 5.2

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

#define NOM_FICHIER "personnes.dat"

#define TAILLE_MAX_NOM 20
#define TAILLE_MAX_PRENOM 15

typedef char Nom[TAILLE_MAX_NOM + 1];
typedef char Prenom[TAILLE_MAX_PRENOM + 1];
typedef unsigned short Age;

typedef struct {
   Nom nom;
   Prenom prenom;
   Age age;
} Personne;

void saisie(char* chaine, size_t taille);
void clear_stdin(void);

int main(void) {
   FILE* fichier = fopen(NOM_FICHIER, "wb");
   if (!fichier) {
      printf("Desole! Le fichier \"%s\" n'a pas pu etre ouvert", NOM_FICHIER);
      return EXIT_FAILURE;
   }

   Personne p;
   printf("--- Pour finir la saisie, donnez un nom 'vide' ---\n");
   do {
      printf("\nNom: ");
      saisie(p.nom, TAILLE_MAX_NOM);
      if (strlen(p.nom) == 0)
         break;
      printf("Prenom : ");
      saisie(p.prenom, TAILLE_MAX_PRENOM);
      printf("Age: ");
      scanf("%hu", &p.age);
      clear_stdin();
      fwrite(&p, sizeof(Personne), 1, fichier);
   } while (1);

   fclose(fichier);
   return EXIT_SUCCESS;
}

void saisie(char* chaine, size_t taille) {
   fgets(chaine, (int) taille + 1, stdin);
   clear_stdin();
   for (size_t i = 0; i < taille; ++i)
      if (chaine[i] == '\n') {
         chaine[i] = '\0';
         break;
      }
}

void clear_stdin(void) {
   fseek(stdin, 0, SEEK_END);
}
```

Exercice 5.3 Lecture intégrale d'un fichier binaire

On suppose disposer du fichier binaire *personnes.dat* créé dans l'exercice précédent.

Ecrire le plus proprement possible un programme C qui recopie le contenu intégral du fichier binaire *personnes.dat* dans un fichier texte *personnes.txt*, histoire de rendre les informations contenues dans le fichier binaire lisibles par un être humain.

Faire en sorte que les données du fichier texte apparaissent alignées en colonnes, comme suit :

```
Nom                 Prenom          Age
Federer             Roger            37
Nadal               Rafael           33
Djokovic            Novak            32
del Potro           Juan Martin      30
```

Solution exercice 5.3

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

#define NOM_FICHIER_BINAIRE "personnes.dat"
#define NOM_FICHIER_TEXTE "personnes.txt"

#define TAILLE_MAX_NOM 20
#define TAILLE_MAX_PRENOM 15

typedef char Nom[TAILLE_MAX_NOM + 1];
typedef char Prenom[TAILLE_MAX_PRENOM + 1];
typedef unsigned short Age;

typedef struct {
   Nom nom;
   Prenom prenom;
   Age age;
} Personne;

int main(void) {
   FILE* fichier_binaire = fopen(NOM_FICHIER_BINAIRE, "rb");
   if (!fichier_binaire) {
      printf("Desole! Le fichier \"%s\" n'a pas pu etre ouvert\n",
             NOM_FICHIER_BINAIRE);
      return EXIT_FAILURE;
   } else {
      FILE* fichier_texte = fopen(NOM_FICHIER_TEXTE, "w");
      if (!fichier_texte) {
         printf("Desole! Le fichier \"%s\" n'a pas pu etre ouvert\n",
                NOM_FICHIER_TEXTE);
         fclose(fichier_binaire);
         return EXIT_FAILURE;
      } else {
         Personne p;
         fprintf(fichier_texte, "%-*s %-*s %s\n",
                 TAILLE_MAX_NOM, "Nom", TAILLE_MAX_PRENOM, "Prenom", "Age");
         while ( fread(&p, sizeof(Personne), 1, fichier_binaire) ) {
            fprintf(fichier_texte, "%-*s %-*s %3hu\n",
                    TAILLE_MAX_NOM, p.nom, TAILLE_MAX_PRENOM, p.prenom, p.age);
         }
         fclose(fichier_binaire);
         fclose(fichier_texte);
         return EXIT_SUCCESS;
      }
   }
}
```

Exercice 5.4 Recherche séquentielle dans un fichier binaire

Ecrire un programme C permettant, à partir du fichier *personnes.dat* créé dans l'exercice 5.2, de retrouver et d'afficher à l'écran les informations relatives à une personne de *nom* donné.

Exemples d'exécution

```
Quel nom recherchez-vous ? : Wawrinka
Desole! Le nom "Wawrinka" ne figure pas dans le fichier

Quel nom recherchez-vous ? : del Potro
Juan Martin del Potro, 30 ans
```

Solution exercice 5.4

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

#define NOM_FICHIER "personnes.dat"

#define TAILLE_MAX_NOM 20
#define TAILLE_MAX_PRENOM 15

typedef char Nom[TAILLE_MAX_NOM + 1];
typedef char Prenom[TAILLE_MAX_PRENOM + 1];
typedef unsigned short Age;

typedef struct {
   Nom nom;
   Prenom prenom;
   Age age;
} Personne;

void lire(char* chaine, size_t taille);

int main(void) {
   FILE* fichier = fopen(NOM_FICHIER, "rb");
   if (!fichier) {
      printf("Desole! Le fichier \"%s\" n'a pas pu etre ouvert\n", NOM_FICHIER);
      return EXIT_FAILURE;
   } else {
      Personne p;
      Nom nomRecherche;
      printf("Quel nom recherchez-vous ? : ");
      lire(nomRecherche, TAILLE_MAX_NOM);
      bool trouve = false;
      while ( ! trouve && fread(&p, sizeof(Personne), 1, fichier) )
         trouve = strcmp(p.nom, nomRecherche) == 0;
      fclose(fichier);
      if (trouve)
         printf("%s %s, %hu ans\n", p.prenom, p.nom, p.age);
      else
         printf("Desole! Le nom \"%s\" ne figure pas dans le fichier\n",
                nomRecherche);
      return EXIT_SUCCESS;
   }
}

void lire(char* chaine, size_t taille) {
   fgets(chaine, (int) taille + 1, stdin);
   fseek(stdin, 0, SEEK_END);
   for (size_t i = 0; i < taille; ++i)
      if (chaine[i] == '\n') {
         chaine[i] = '\0';
         break;
      }
}
```

Exercice 5.5 Recherche par accès direct dans un fichier binaire

En exploitant l'accès direct, écrire un programme C permettant, à partir du fichier *personnes.dat* créé dans l'exercice 5.2, de retrouver et d'afficher à l'écran les informations relatives à une personne de *rang*[1] donné.

[1] Convention : La première personne stockée dans le fichier a pour rang 1, la deuxième, le rang 2, etc.

Exemples d'exécution

```
Quel rang recherchez-vous ? : 0
Le fichier contient 4 enregistrement(s).
Desole! Aucune personne ayant ce rang ne figure dans le fichier.

Quel rang recherchez-vous ? : 1
Le fichier contient 4 enregistrement(s).
La personne de rang 1 est : Roger Federer, 37 ans

Quel rang recherchez-vous ? : 4
Le fichier contient 4 enregistrement(s).
La personne de rang 4 est : Juan Martin del Potro, 30 ans

Quel rang recherchez-vous ? : 5
Le fichier contient 4 enregistrement(s).
Desole! Aucune personne ayant ce rang ne figure dans le fichier.
```

Solution exercice 5.5

```c
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>

#define NOM_FICHIER "personnes.dat"

#define TAILLE_MAX_NOM 20
#define TAILLE_MAX_PRENOM 15

typedef char Nom[TAILLE_MAX_NOM + 1];
typedef char Prenom[TAILLE_MAX_PRENOM + 1];
typedef unsigned short Age;

typedef struct {
   Nom nom;
   Prenom prenom;
   Age age;
} Personne;

int main(void) {
   FILE* fichier = fopen(NOM_FICHIER, "rb"); // accès direct en lecture seule
   size_t nbEnregistrements;
   int rang; // rang de la personne recherche dans le fichier

   if (!fichier) {
      printf("Desole! Le fichier \"%s\" n'a pas pu etre ouvert\n", NOM_FICHIER);
      return EXIT_FAILURE;
   } else {
      // Déterminer combien d'enregistrements contient le fichier
      fseek(fichier, 0, SEEK_END);
      nbEnregistrements = (size_t)ftell(fichier) / sizeof(Personne);
      if (nbEnregistrements == 0)
         printf("Le fichier \"%s\" est vide\n", NOM_FICHIER);
      else {
         printf("Quel rang recherchez-vous ? : ");
         scanf("%d", &rang);
         printf("Le fichier contient %" PRIuMAX " enregistrement(s).\n",
                (uintmax_t) nbEnregistrements);
         if (rang <= 0 || (size_t) rang > nbEnregistrements)
            printf("Desole! Aucune personne ayant ce rang "
                   "ne figure dans le fichier.\n");
         else {
            Personne p;
            // Se positionner sur l'enregistrement concerné
            fseek(fichier, (rang - 1) * (int) sizeof(Personne), SEEK_SET);
            // Lire les infos correspondantes
            fread(&p, sizeof(Personne), 1, fichier);
            // Afficher le résultat à l'écran
            printf("La personne de rang %d est : ", rang);
            printf("%s %s, %hu ans\n", p.prenom, p.nom, p.age);
         }
      }
      fclose(fichier);
      return EXIT_SUCCESS;
   }
}
```

Exercice 5.6 Compteurs d'octets

Ecrire un programme complet permettant d'ouvrir n'importe quel fichier (image, son, document, …) et de compter le nombre d'octets de chaque valeur (0 à 255) figurant dans ce fichier.

Le programme doit demander à l'utilisateur le nom du fichier (nom pouvant éventuellement contenir des espaces) à traiter. La saisie du nom doit être sécurisée (pas de débordement de buffer possible). Le nom du fichier (extension incluse) doit être plus petit ou égal à 30 caractères.

Avant de terminer le programme, il faut afficher le résultat à l'écran (compteurs de chaque valeur d'octet).

Solution exercice 5.6

```
#include <inttypes.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

#define TAILLE_MAX_NOM_FICHIER 30

typedef unsigned char uchar;

void lire(char* chaine, size_t taille);

int main(void) {
   char nomFichier[TAILLE_MAX_NOM_FICHIER + 1];
   FILE* f;
   printf("Entrez le nom du fichier a lire (%d caract. max) > ",
          TAILLE_MAX_NOM_FICHIER);
   lire(nomFichier, TAILLE_MAX_NOM_FICHIER);
   f = fopen(nomFichier, "rb");
   if (!f) {
      printf("Desole! Le fichier \"%s\" n'a pas pu etre ouvert\n", nomFichier);
      return EXIT_FAILURE;
   } else {
      fseek(f, 0, SEEK_END);
      const size_t TAILLE_FICHIER = (size_t) ftell(f);
      fseek(f, 0, SEEK_SET);
      uchar contenuFichier[TAILLE_FICHIER];
      if ( fread(contenuFichier, sizeof(uchar), TAILLE_FICHIER, f)
           != TAILLE_FICHIER ) {
         printf("Une erreur s'est produite en cours de lecture!\n");
         fclose(f);
         return EXIT_FAILURE;
      } else {
         size_t compteurs[UCHAR_MAX + 1] = {0};
         for (size_t i = 0; i < TAILLE_FICHIER; ++i)
            compteurs[contenuFichier[i]]++;
         for (int i = 0; i < UCHAR_MAX + 1; ++i)
            printf("%d : %" PRIuMAX "\n", i, (uintmax_t) compteurs[i]);
         fclose(f);
         return EXIT_SUCCESS;
      }
   }
}

void lire(char* chaine, size_t taille) {
   fgets(chaine, (int) taille + 1, stdin);
   fseek(stdin, 0, SEEK_END);
   for (size_t i = 0; i < taille; ++i)
      if (chaine[i] == '\n') {
         chaine[i] = '\0';
         break;
      }
}
```

Exercice 5.7 Lecture d'un fichier texte sans utiliser de boucle

Ecrire un programme C permettant d'afficher à l'écran l'intégralité d'un fichier texte donné, sans jamais utiliser de boucle.

Solution exercice 5.7

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

#define NOM_FICHIER "Ex5-7.c"

int main(void) {

   FILE* f = fopen(NOM_FICHIER, "r");

   if (!f) { // Si ouverture du fichier impossible
      printf("Ouverture du fichier \"%s\" impossible.\n", NOM_FICHIER);
      return EXIT_FAILURE;
   } else {
      // Récupérer la taille en octets du fichier
      fseek(f, 0, SEEK_END);
      const size_t NB_OCTETS = (size_t) ftell(f);
      // Créer dynamiquement un buffer dans lequel sera stocké le contenu
      // du fichier
      char* buffer = (char*) calloc(NB_OCTETS + 1, sizeof(char));
      if (!buffer) {
         printf("Memoire insuffisante pour creer le buffer.\n");
         fclose(f);
         return EXIT_FAILURE;
      } else {
         // Se repositionner au début du fichier
         rewind(f); // ou fseek(f, 0, SEEK_SET);
         // Lire le contenu du fichier d'un seul tenant et le stocker dans buffer
         fread(buffer, NB_OCTETS, 1, f);
         // Afficher buffer à l'écran
         printf("%s\n\n", buffer);
         // Récupérer la mémoire allouée pour buffer
         free(buffer);
         fclose(f);
         return EXIT_SUCCESS;
      }
   }
}
```

Exercice 5.8 Heures d'ensoleillement

On suppose disposer du fichier *xxx.input* suivant :
(xxx est variable; l'extension .input est fixe)

```
Heures  Lieu
--------------------
1466    Aarau
1545    Adelboden
1637    Bale
1682    Berne
1350    Engelberg
1795    Evolene
1828    Geneve
1570    Interlaken
1832    Jungfraujoch
1872    Lausanne
1424    Lucerne
2143    Montana
1641    Neuchatel
1844    Nyon
1719    Payerne
1809    Santis
1448    Schaffhouse
2094    Sion
1595    Wadenswil
1685    Zermatt
1566    Zurich
```

Ce fichier fournit le nombre d'heures annuelles d'ensoleillement de divers lieux en Suisse.

Ecrire un programme C permettant de produire, à partir du fichier *xxx.input*, le fichier *xxx.output* suivant :

```
Moyenne des heures d'ensoleillement : 1692.6

Lieux ayant plus d'heures d'ensoleillement que la moyenne :
- Evolene (1795)
- Geneve (1828)
- Jungfraujoch (1832)
- Lausanne (1872)
- Montana (2143)
- Nyon (1844)
- Payerne (1719)
- Santis (1809)
- Sion (2094)
```

Contraintes

- Le nom du fichier d'input est un paramètre de la ligne de commande
- La longueur d'une ligne du fichier d'input est de 30 caractères maximum
- Le fichier d'input contient au maximum 100 lignes de données heures – lieu
- Le nom du fichier d'output est de 50 caractères maximum (extension *.output* incluse)
- La taille maximale d'une ligne du fichier d'input ainsi que le nombre maximal de données heures – lieu sont des points garantis (peuvent être considérés comme toujours vrais)

Précision sur le comportement du programme

- Si le nom du fichier d'input n'est pas passé en paramètre de la ligne de commande,
  - le programme le signale via le message :
    `Entrez SVP le nom du fichier a lire (max. n caract.).`
  - le programme se termine
- Si le nom du fichier d'input est passé en paramètre de la ligne de commande, mais qu'il est trop long,
  - le programme le signale via le message :
    `Le nom du fichier est trop long. Il doit comporter au max. n caracteres.`
  - le programme se termine
- S'il s'avère impossible d'ouvrir le fichier d'input ou le fichier d'output,
  - le programme le signale via le message :
    `Impossible d'ouvrir le fichier "xxx.input" (ou "xxx.output").`
  - le programme se termine

Solution exercice 5.8

N.B. L'extension .input contient un caractère de moins que l'extension .output

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

#define TAILLE_MAX_NOM_FICHIER 50
#define TAILLE_MAX_LIGNE_FICHIER 30
#define NB_MAX_LIGNES 100

int main(int argc, char* argv[]) {

   FILE *fichier_in,
        *fichier_out;

   // Quitter si le nom du fichier à lire n'a pas été passé en paramètre
   // de la ligne de commande
   if (argc != 2) {
      printf("Entrez SVP le nom du fichier a lire (max. %" PRIuMAX " caract.).\n",
             (uintmax_t) (TAILLE_MAX_NOM_FICHIER - 1));
      return EXIT_FAILURE;
   }

   // Quitter si le nom du fichier à lire passé en paramètre de la ligne de
   // commande est trop long
   if (strlen(argv[1]) > TAILLE_MAX_NOM_FICHIER - 1) {
      printf("Le nom du fichier est trop long."
             " Il doit comporter au max. %" PRIuMAX " caracteres.\n",
             (uintmax_t) (TAILLE_MAX_NOM_FICHIER - 1));
      return EXIT_FAILURE;
   }

   // Quitter si la tentative d'ouverture du fichier d'input échoue.
   fichier_in = fopen(argv[1], "r");
   if (fichier_in == NULL) {
      printf("Impossible d'ouvrir le fichier \"%s\".", argv[1]);
      return EXIT_FAILURE;
   }
```

```
    // Construction du nom du fichier d'output
    char nom_fichier_out[TAILLE_MAX_NOM_FICHIER + 1] = "";
    strncpy(nom_fichier_out, argv[1], strlen(argv[1]) - strlen(".input"));
    strcat(nom_fichier_out, ".output");

    // Quitter si la tentative d'ouverture du fichier d'output échoue.
    fichier_out = fopen(nom_fichier_out, "w");
    if (fichier_out == NULL) {
       printf("Impossible d'ouvrir le fichier \"%s\".", nom_fichier_out);
       // Fermer le fichier d'input préalablement ouvert
       fclose(fichier_in);
       return EXIT_FAILURE;
    }

    // "Sauter" les 2 premières lignes du fichier d'input
    char ligne[TAILLE_MAX_LIGNE_FICHIER + 1];
    fgets(ligne, TAILLE_MAX_LIGNE_FICHIER + 1, fichier_in);
    fgets(ligne, TAILLE_MAX_LIGNE_FICHIER + 1, fichier_in);

    // Lire et stocker les données statistiques
    char lieux[NB_MAX_LIGNES][TAILLE_MAX_LIGNE_FICHIER + 1];
    unsigned heures_ensoleillement[NB_MAX_LIGNES];
    unsigned total_heures_ensoleillement = 0;

    size_t i = 0;
    while (fscanf(fichier_in, "%u %s",
                  &heures_ensoleillement[i],&lieux[i][0]) == 2) {
       total_heures_ensoleillement += heures_ensoleillement[i];
       ++i;
    }

    // Calcul du nb d'heures d'ensoleillement moyen
    double moyenne_heures_ensoleillement = 0;
    if (i != 0) {
       moyenne_heures_ensoleillement = total_heures_ensoleillement / (double) i;
    }

    // Ecriture des résultats dans le fichier d'output
    fprintf(fichier_out, "Moyenne des heures d'ensoleillement : %.1f\n\n",
            moyenne_heures_ensoleillement);
    fprintf(fichier_out,
            "Lieux ayant plus d'heures d'ensoleillement que la moyenne :\n");
    for (size_t j = 0; j < i; ++j) {
       if (heures_ensoleillement[j] > moyenne_heures_ensoleillement) {
          fprintf(fichier_out, "- %s (%u)\n", lieux[j], heures_ensoleillement[j]);
       }
    }

    // Fermeture des fichiers
    fclose(fichier_in);
    fclose(fichier_out);

    return EXIT_SUCCESS;

}
```