Solution exercice 4.2

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

#define PRINT(STR) printf(#STR " = %s\n", STR)

char* strcpy(char* to, const char* from);

int main(void) {

   char s1[10];
   char* s2;
   const char* s3;

   s2 = strcpy(s1, "");
   PRINT(s1);
   PRINT(s2);

   strcpy(s1, "ABC");
   PRINT(s1);
   PRINT(s2);

   s3 = "DEF";
   strcpy(s1, s3);
   PRINT(s1);

   strcpy(s1, strcpy(s1, "ABC"));
   PRINT(s1);

//    char* msg = "ABC";
//    strcpy(msg, ""); // provoquerait un crash à l'exécution car msg pointe
//                     // sur chaîne constante ("ABC"), donc non modifiable

   return EXIT_SUCCESS;
}

char* strcpy(char* to, const char* from) {
   char* tmp = to;
   while ((*to++ = *from++) != '\0');
   return tmp;
}

// s1 =
// s2 =
// s1 = ABC
// s2 = ABC
// s1 = DEF
// s1 = ABC
```

Solution exercice 4.3

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

#define PRINT(STR) printf(#STR " = %s\n", STR)

char* strncpy(char* to, const char* from, size_t size);

int main(void) {

   {
      const char* from = "AB";
      char to[] = "XXXXXX";
      for (size_t i = 0; i <= 3; ++i) {
         strncpy(to, from, i);
         PRINT(to);
      }
   }

   {
      const char* from = "AB";
      char to[] = "XXXXXX";
      const size_t TAILLE = 6;
      strncpy(to, from, 4);
      for (size_t i = 0; i <= TAILLE; ++i)
         printf("%d ", (int)to[i]);
      printf("\n");
   }

   return EXIT_SUCCESS;
}

char* strncpy(char* to, const char* from, size_t size) {
   size_t i;
   for (i = 0; i < size && from[i] != '\0'; ++i)
      to[i] = from[i];
   for (; i < size; ++i)
      to[i] = '\0';
   return to;
}

// char* strncpy(char* dest, const char* src, size_t num) {
//    char* ret = dest;
//    do {
//       if (!num--)
//          return ret;
//     } while ((*dest++ = *src++));
//     while (num--)
//       *dest++ = '\0';
//     return ret;
// }

// to = XXXXXX
// to = AXXXXX
// to = ABXXXX
// to = AB
// 65 66 0 0 88 88 0
```

Solution exercice 4.4

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

#define PRINT(STR) printf(#STR " = %s\n", STR)

char* strcat(char* to, const char* from);

int main(void) {

   char to[10] = ""; // nécessaire pour que le premier caractère de to soit '\0'.
                     // Aussi possible d'écrire {'\0'}
   const char* from = "ABC";
   char* s;

   strcat(to, from);
   PRINT(to);

   s = strcat(to, "DEF");
   PRINT(to);
   PRINT(s);

   return EXIT_SUCCESS;
}

char* strcat(char* to, const char* from) {
   char* tmp = to;
   while (*to) to++; // se positionner sur '\0'
   while ((*to++ = *from++) != '\0'); // idem strcpy
   return tmp;
}

// to = ABC
// to = ABCDEF
// s = ABCDEF
```

Solution exercice 4.5

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

#define PRINT(STR) printf(#STR " = %s\n", STR)

char* strncat(char* to, const char* from, size_t size);

int main(void) {

   char to[10] = ""; // nécessaire pour que le premier caractère de to soit '\0'.
                     // Aussi possible d'écrire {'\0'}
   const char* from = "ABC";

   for (size_t i = 1; i <= 4; ++i) {
      strncat(to, from, i);
      PRINT(to);
   }

   return EXIT_SUCCESS;
}

char* strncat(char* to, const char* from, size_t size) {
   char* tmp = to;
   if (size) {
      while (*to) to++;
      while ( (*to++ = *from++) != '\0' ) {
         if (--size == 0) {
            *to = '\0';
            break;
         }
      }
   }
   return tmp;
}

// to = A
// to = AAB
// to = AABABC
// to = AABABCABC
```

Solution exercice 4.9

```
#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