```
 1 # `9-more-pointers`
 2
 3 ## `strlen.c`
 4
 5 - C string literal
 6 - `while (str[len++] != '\0')` vs.
 7   `while (++str[len] != '\0')` vs.
 8   `while (++str[len])`
 9 - `\0` vs. `0`
10
11 ## `strcpy.c`
12
13 ## `strcmp.c`
```

```c
//
// Created by hfwei on 2023/11/30.
// Visualization:https://pythontutor.com/render.html#code=//%0A//%20Created%20by%20hfwei%20on%202023/11/30.%0A//%0A%0A%23include%20%3Cstdio.h%3E%0A%0Aint%20main%28void%29%20%7B%0A%20%20char%20msg%5B%5D%20%3D%20%22Hello%20World!%22%3B%0A%20%20printf%28%22%25s%5Cn%22,%20msg%29%3B%0A%0A%20%20msg%5B0%5D%20%3D%20'N'%3B%0A%20%20printf%28%22%25s%5Cn%22,%20msg%29%3B%0A%0A%20%20//%20string%20literal%3B%0A%20%20//%20may%20be%20stored%20in%20read-only%20memory%0A%20%20char%20*ptr_msg%20%3D%20%22Hello%20World!%22%3B%0A%0A%20%20//%20undefined%20behavior%0A%0A%20%20//%20On%20Linux%20%28or%20Debug%20mode%20on%20Windows%29%3A%20interrupted%20by%20signal%2011%3A%20SIGSEGV%0A%20%20//%20SIG%3A%20signal%3B%20SEGV%3A%20segmentation%20violation%0A%0A%20%20//%20On%20Windows%3A%20Process%20finished%20with%20exit%20code%20-1073741819%20%280xC0000005%29%0A%20%20//%20See%20https%3A//learn.microsoft.com/en-us/openspecs/windows_protocols/ms-erref/596a1078-e883-4972-9bbc-49e60bebca55%0A%20%20ptr_msg%5B0%5D%20%3D%20'N'%3B%0A%20%20printf%28%22%25s%5Cn%22,%20msg%29%3B%0A%0A%20%20return%200%3B%0A%7D&cppShowMemAddrs=true&cumulative=true&curInstr=7&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false
// See String literals: https://en.cppreference.com/w/c/language/string_literal
//

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

int main(void) {
  // it is actually char msg[] = { 'H', 'e', ..., '\0' };
  char msg[] = "Hello World!";
  printf("%s\n", msg);

  // array name is read-only
  // msg = "Hello";

  msg[0] = 'N';
  printf("%s\n", msg);

  // string literal;
  // may be stored in read-only memory
  char *ptr_msg = "Hello World!";

  // undefined behavior

  // On Linux (or Debug mode on Windows): interrupted by signal 11: SIGSEGV
  // SIG: signal; SEGV: segmentation violation

  // On Windows: Process finished with exit code -1073741819 (0xC0000005)
  // See https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-erref/596a1078-e883-4972-9bbc-49e60bebca55
```

```c
  ptr_msg[0] = 'N';
  printf("%s\n", msg);

  // using malloc
  char *malloc_msg = malloc(20);
  malloc_msg = "Hello World!";
  malloc_msg[0] = 'N';

  return 0;
}
```

```c
//
// file: strlen.c
// Created by hfwei on 2023/11/29.
// See https://en.cppreference.com/w/c/string/byte/strlen
//

#include <stdio.h>

int StrLen(const char *s);
int StrLen1(const char *s);
int StrLen2(const char *s);
//  The behavior is undefined if str is not a pointer to a null-terminated byte string.
size_t StrLenStd(const char *s);

int main() {
  char msg[] = "Hello World!";

  printf("StrLen(%s) = %d\n", msg, StrLen(msg));
  printf("StrLenStd(%s) = %zu\n", msg, StrLenStd(msg));

  return 0;
}

int StrLen(const char *s) {
  int len = 0;
  while (s[len] != '\0') {
    len++;
  }

  return len;
}

int StrLen1(const char *s) {
  int len = 0;
  while (s[len++] != '\0');

  return len - 1;
}

int StrLen2(const char *s) {
  int len = -1;
  while (s[++len] != '\0');

  return len;
}

size_t StrLenStd(const char *s) {
  const char *sc;
  for (sc = s; *sc != '\0'; sc++);

  return sc - s;
}
```

```c
//
// file: strcpy.c
// 7 versions of strcpy
// Created by hfwei on 2022/11/29.
//
// C Operator Precedence: https://en.cppreference.com/w/c/language/operator_precedence#:~:text=C%20Operator%20Precedence%20%20%20%20Precedence%20,union%20member%20access%20%2028%20more%20rows%20
//

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

/**
 * @brief We assume that there is enough room for storing src.
 *  Otherwise, it is an undefined behavior.
 *
 *  If copying takes place between objects that overlap,
 *  then behavior is undefined.
 *
 *  In Docs:
 *  (1) The behavior is undefined if the dest array is not large enough.
 *  (2) The behavior is undefined if the strings overlap.
 *  (3) The behavior is undefined if either dest is not a pointer to a character array
 *      or src is not a pointer to a null-terminated byte string.
 *
 * @param dest may NOT be null-terminated
 * @param src  must be null-terminated
 */
void StrCpy(char *dest, const char *src);
void StrCpy1(char *dest, const char *src);
void StrCpy2(char *dest, const char *src);
void StrCpy3(char *dest, const char *src);
void StrCpy4(char *dest, const char *src);
void StrCpy5(char *dest, const char *src);
char *StrCpyStd(char *dest, const char *src);

int main() {
  const char *src = "Hello World";
  // VLA (Do not use it; it is optional since C11)
  // char dest[strlen(src) + 1];
  char *dest = malloc(strlen(src) + 1);

  StrCpy4(dest, src);
  strlen(dest);
  printf("dest = %s\n", dest);

  strlen(StrCpyStd(dest, src));

  return 0;
```

```c
}

void StrCpy(char *dest, const char *src) {
  int i = 0;
  while (src[i] != '\0') {
    dest[i] = src[i];
    i++;
  }

  dest[i] = '\0';
}

void StrCpy1(char *dest, const char *src) {
  int i = 0;
  while ((dest[i] = src[i]) != '\0') {
    i++;
  }
}

void StrCpy2(char *dest, const char *src) {
  int i = 0;
  // dest[i] : *(dest + i)
  while ((*(dest + i) = *(src + i)) != '\0') {
    i++;
  }
}

void StrCpy3(char *dest, const char *src) {
  while ((*dest = *src) != '\0') {
    src++;
    dest++;
  }

  printf("%s\n", src);
}

// See C Operator Precedence: https://en.cppreference.com/w/c/language/operator_precedence#:~:text=C%20Operator%20Precedence%20%20%20%20Precedence%20,union%20member%20access%20%2028%20more%20rows%20
// Visualization: https://pythontutor.com/render.html#code=%23include%20%3Cstring.h%3E%0A%23include%20%3Cstdio.h%3E%0A%23include%20%3Cstdlib.h%3E%0A%0Avoid%20StrCpy4%28char%20*dest,%20const%20char%20*src%29%3B%0A%0Aint%20main%28%29%20%7B%0A%20%20const%20char%20*src%20%3D%20%22Hello%20World%22%3B%0A%20%20char%20*dest%20%3D%20malloc%28strlen%28src%29%20%2B%201%29%3B%0A%0A%20%20StrCpy4%28dest,%20src%29%3B%0A%20%20printf%28%22dest%20%3D%20%25s%5Cn%22,%20dest%29%3B%0A%0A%20%20return%200%3B%0A%7D%0A%0A%0Avoid%20StrCpy4%28char%20*dest,%20const%20char%20*src%29%20%7B%0A%20%20//%20dest%2B%2B%3A%20dest,%20dest%20%3D%20dest%20%2B%201%0A%20%20//%20*dest%2B%2B%3A%20*dest,%20not%20*%28dest%20%2B%201%29%0A%20%20while%20%28%0A%20%20%20%20%28*dest%2B%2B%20%0A%20%20%20%20%20%20%3D%20*src%2B%2B%29%20%0A%20%20%20%20%20%20%20%20%20%20%20%20!%3D%20'%5C0'%29%3B%0A%7D&cppShowMemAddrs=true&cumulative=true&curInstr=0&heapPrimitives=nevernest&mode=display&
```

```c
// origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false
// Tricky difference between StrCpy3: src, dest beyond '\0'
// You SHOULD be able to understand this!!!
void StrCpy4(char *dest, const char *src) {
  // dest++: dest, dest = dest + 1
  // *dest++: *dest, not *(dest + 1)
  while ((*dest++ = *src++) != '\0');

  printf("%s\n", src);
}

// NOT Recommended!
// See ASCII Chart: https://en.cppreference.com/w/c/language/ascii
void StrCpy5(char *dest, const char *src) {
  // '\0': null character (NUL), 0
  // `\0' is not NULL
  while ((*dest++ = *src++));
}

// See https://en.cppreference.com/w/c/string/byte/strcpy
char *StrCpyStd(char *dest, const char *src) {
  for (char *s = dest; (*s++ = *src++) != '\0';);
  return dest;
}
```

```c
//
// file: strcmp.c
// Created by hfwei on 2022/11/29.
//

#include <stdio.h>

int StrCmp(const char *s1, const char *s2);
int StrCmpStd(const char *s1, const char *s2);

int StrNCmpStd(const char *s1, const char *s2, int n);

int main() {
  const char *str1 = "hi, C";
  const char *str2 = "hi, c";

  printf("StrCmp(\"%s\", \"%s\") = %d\n",
         str1, str2, StrCmp(str1, str2));
  // printf("StrCmpStd(\"%s\", \"%s\") = %d\n",
  //        str1, str2, StrCmpStd(str1, str2));
  //
  // int n = 2;
  // printf("StrNCmp(\"%s\", \"%s\", %d) = %d\n",
  //        str1, str2, n, StrNCmp(str1, str2, n));

  return 0;
}

// Wrong Version
// hi vs. hi ('\0')
// int StrCmp(const char *s1, const char *s2) {
//   while (*s1 == *s2) {
//     s1++;
//     s2++;
//   }
//
//   return *s1 - *s2;
// }

//
int StrCmp(const char *s1, const char *s2) {
  while (*s1 == *s2 && (*s1 != '\0' && *s2 != '\0')) {
    s1++;
    s2++;
  }

  if (*s1 == '\0' && *s2 == '\0') {
    return 0;
  }

  // char: unsigned char, signed char
  return (*(const unsigned char *) s1)
             < (*(const unsigned char *) s2) ? -1 : 1;
```

```c
}

// See https://en.cppreference.com/w/c/string/byte/strcmp
//
// Compares two null-terminated byte strings lexicographically.
// The sign of the result is the sign of the difference between the values of the first pair of characters (both interpreted as unsigned char) that differ in the strings being compared.
// The behavior is undefined if lhs or rhs are not pointers to null-terminated byte strings.
int StrCmpStd(const char *s1, const char *s2) {
  for (; *s1 == *s2; s1++, s2++) {
    if (*s1 == '\0') {
      return 0;
    }
  }

  return (*(const unsigned char *) s1)
              < (*(const unsigned char *) s2) ? -1 : 1;
}

// See https://en.cppreference.com/w/c/string/byte/strncmp
// Compares at most count characters of two possibly null-terminated arrays.
// The comparison is done lexicographically. Characters following the null character are not compared.
int StrNCmpStd(const char *s1, const char *s2, int n) {
  for (; 0 < n; n--, s1++, s2++) {
    if (*s1 != *s2) {
      return (*(const unsigned char *) s1)
                  < (*(const unsigned char *) s2) ? -1 : 1;
    } else if (*s1 == '\0') { // *s1 == *s2 == '\0'
      return 0;
    }
  }

  return 0;
}
```