```c
// Created by hfwei on 2024/11/20.

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

char *StrCat(char *s1, const char *s2);
char *StrCatGLibC(char *dest, const char *src);

char *StrNCat(char *s1, const char *s2, size_t n);
size_t StrNLen(const char *s, size_t max);
char *StrNCatGLic(char *s1, const char *s2, size_t n);

int main(void) {
  char str[50] = "Hello ";
  char str2[50] = "World!";

//  strcat(str, str2);
//  strcat(str, " ...");
//  strcat(str, " Goodbye World!");

  strcat(strcat(strcat(str, str2), " ..."), " Goodbye World!");

//  strcat(str, str2);
//  strncat(str, " Goodbye World!", 3);

  puts(str);

  return 0;
}

char *StrCat(char *s1, const char *s2) {
  char *s;
  for (s = s1; *s != '\0'; s++);

  for (; (*s = *s2) != '\0'; s++, s2++);

  return s1;
}

// https://github.com/bminor/glibc/blob/master/string/strcat.c
char *StrCatGLibC(char *dest, const char *src) {
  strcpy(dest + strlen(dest), src);
```

```c
  return dest;
}

char *StrNCat(char *s1, const char *s2, size_t n) {
  char *s;
  for (s = s1; *s != '\0'; s++);

  for (; 0 < n && *s2 != '\0'; --n) {
    *s++ = *s2++;
  }
  *s = '\0';

  return s1;
}

// https://github.com/intel/safestringlib/blob/master/safeclib/strnlen_s.c
size_t StrNLen(const char *s, size_t max) {
  size_t count = 0;

  while (max && *s) {
    count++;
    max--;
    s++;
  }

  return count;
}

char *StrNCatGLic(char *s1, const char *s2, size_t n) {
  char *s = s1;

  s1 += strlen(s1);

  size_t ss = StrNLen(s2, n);

  s1[ss] = '\0';
  memcpy(s1, s2, ss);

  return s;
}
```

```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 StrCmpGLibC(const char *p1, const char *p2);

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++;
```

```c
    s2++;
  }

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

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

// 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
```

```
 79 unsigned char *) s2) ? -1 : 1;
 80     } else if (*s1 == '\0') { // *s1 == *s2 == '\0'
 81       return 0;
 82     }
 83   }
 84
 85   return 0;
 86 }
 87
 88 int StrCmpGLibC(const char *p1, const char *p2) {
 89   const unsigned char *s1 = (const unsigned char *) p1;
 90   const unsigned char *s2 = (const unsigned char *) p2;
 91   unsigned char c1;
 92   unsigned char c2;
 93
 94   do {
 95     c1 = *s1++;
 96     c2 = *s2++;
 97     if (c1 == '\0')
 98       return c1 - c2;
 99   } while (c1 == c2);
100
101   return c1 - c2;
102 }
```

```
1 // file: strcpy.c
2 // 7 versions of strcpy
3 // Created by hfwei on 2022/11/29.
4 //
5 // C Operator Precedence:
6 // 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
7 // Python Tutor:
8 // https://pythontutor.com/render.html#code=%23include%20%
  3Cstdio.h%3E%0A%23include%20%3Cstdlib.h%3E%0A%23include%20
  %3Cstring.h%3E%0A%0Avoid%20StrCpy%28char%20*dest,%20const%
  20char%20*src%29%3B%0Avoid%20StrCpy1%28char%20*dest,%
  20const%20char%20*src%29%3B%0Avoid%20StrCpy2%28char%20*
  dest,%20const%20char%20*src%29%3B%0Avoid%20StrCpy3%28char%
  20*dest,%20const%20char%20*src%29%3B%0Avoid%20StrCpy4%
  28char%20*dest,%20const%20char%20*src%29%3B%0Avoid%
  20StrCpy5%28char%20*dest,%20const%20char%20*src%29%3B%
  0Achar%20*StrCpyStd%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%20strlen%
  28StrCpyStd%28dest,%20src%29%29%3B%0A%0A%20%20return%200%
  3B%0A%7D%0A%0Avoid%20StrCpy%28char%20*dest,%20const%20char
  %20*src%29%20%7B%0A%20%20int%20i%20%3D%200%3B%0A%20%
  20while%20%28src%5Bi%5D%20!%3D%20'%5C0'%29%20%7B%0A%20%20%
  20%20dest%5Bi%5D%20%3D%20src%5Bi%5D%3B%0A%20%20%20%20i%2B%
  2B%3B%0A%20%20%7D%0A%0A%20%20dest%5Bi%5D%20%3D%20'%5C0'%3B
  %0A%7D%0A%0Avoid%20StrCpy1%28char%20*dest,%20const%20char%
  20*src%29%20%7B%0A%20%20int%20i%20%3D%200%3B%0A%20%20while
  %20%28%28dest%5Bi%5D%20%3D%20src%5Bi%5D%29%20!%3D%20'%5C0
  '%29%20%7B%0A%20%20%20%20i%2B%2B%3B%0A%20%20%7D%0A%7D%0A%
  0Avoid%20StrCpy2%28char%20*dest,%20const%20char%20*src%29%
  20%7B%0A%20%20int%20i%20%3D%200%3B%0A%20%20while%20%28%28
  *%28dest%20%2B%20i%29%20%3D%20*%28src%20%2B%20i%29%29%20!%
  3D%20'%5C0'%29%20%7B%0A%20%20%20%20i%2B%2B%3B%0A%20%20%7D%
  0A%7D%0A%0Avoid%20StrCpy3%28char%20*dest,%20const%20char%
  20*src%29%20%7B%0A%20%20while%20%28%28*dest%20%3D%20*src%
  29%20!%3D%20'%5C0'%29%20%7B%0A%20%20%20%20src%2B%2B%3B%0A%
  20%20%20%20dest%2B%2B%3B%0A%20%20%7D%0A%0A%20%20printf%28%
  22%25s%5Cn%22,%20src%29%3B%0A%7D%0A%0Avoid%20StrCpy4%
```

```
 8 28char%20*dest,%20const%20char%20*src%29%20%7B%0A%20%
   20while%20%28%28*dest%2B%2B%20%3D%20*src%2B%2B%29%20!%3D%
   20'%5C0'%29%3B%0A%0A%20%20printf%28%22%25s%5Cn%22,%20src%
   29%3B%0A%7D%0A%0Avoid%20StrCpy5%28char%20*dest,%20const%
   20char%20*src%29%20%7B%0A%20%20while%20%28%28*dest%2B%2B%
   20%3D%20*src%2B%2B%29%29%0A%20%20%20%20%3B%0A%7D%0A%0Achar
   %20*StrCpyStd%28char%20*dest,%20const%20char%20*src%29%20%
   7B%0A%20%20for%20%28char%20*s%20%3D%20dest%3B%20%28*s%2B%
   2B%20%3D%20*src%2B%2B%29%20!%3D%20'%5C0'%3B%29%3B%0A%20%
   20return%20dest%3B%0A%7D&cumulative=true&curInstr=0&
   heapPrimitives=nevernest&mode=display&origin=opt-frontend.
   js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=
   false
 9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13
14 /**
15  * @brief We assume that there is enough room for storing
   src.
16  *  Otherwise, it is an undefined behavior.
17  *
18  *  If copying takes place between objects that overlap,
19  *  then behavior is undefined.
20  *
21  *  In Docs:
22  *  (1) The behavior is undefined if the dest array is not
    large enough.
23  *  (2) The behavior is undefined if the strings overlap.
24  *  (3) The behavior is undefined if either dest is not a
   pointer to a character
25  * array or src is not a pointer to a null-terminated byte
    string.
26  *
27  * @param dest may NOT be null-terminated
28  * @param src  must be null-terminated
29  */
30 void StrCpy1(char *dest, const char *src);
31 void StrCpy2(char *dest, const char *src);
32 void StrCpy3(char *dest, const char *src);
33 void StrCpy4(char *dest, const char *src);
34 void StrCpy5(char *dest, const char *src);
35 void StrCpy6(char *dest, const char *src);
```

```c
char *StrCpyStd(char *dest, const char *src);
char *StrNCpyStd(char *dest, const char *src, size_t n) {

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);

  StrCpy5(dest, src);
  strlen(dest);
  printf("dest = %s\n", dest);

  strlen(StrCpyStd(dest, src));

  return 0;
}

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

  dest[i] = '\0';
}

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

void StrCpy3(char *dest, const char *src) {
  int i = 0;
  // dest[i] : *(dest + i)
  while ((*(dest + i) = *(src + i)) != '\0') {
    i++;
  }
}

void StrCpy4(char *dest, const char *src) {
```

```c
  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%21Operator%20Precedence%20%20%20%20Precedence%20,union%20member%20access%20%2028%20more%20rows%20
// Visualization:
// https://pythontutor.com/visualize.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%20free%28dest%29%3B%0A%20%20%0A%20%20return%200%3B%0A%7D%0A%0A%0Avoid%20StrCpy4%28char%20*dest,%20const%20char%20*src%29%20%7B%0A%20%20while%20%28%0A%20%20%20%20%28*dest%2B%2B%20%0A%20%20%20%20%20%3D%20*src%2B%2B%29%20%0A%20%20%20%20%20%20%20%20!%3D%20'%5C0'%29%3B%0A%0A%20%20printf%28%22%25s%5Cn%22,%20src%29%3B%0A%7D&cumulative=true&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false
// Tricky difference between StrCpy4: src, dest beyond '\0'
// You SHOULD be able to understand this!!!
void StrCpy5(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/
```

```c
language/ascii
void StrCpy6(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;
}

char *StrNCpyStd(char *dest, const char *src, size_t n) {
  char *s;
  for (s = dest; 0 < n && *src != '\0'; --n) {
    *s++ = *src++;
  }

  for (; 0 < n; --n) {
    *s++ = '\0';
  }

  return dest;
}
```

```c
// file: strlen.c
// Created by hfwei on 2024/11/20.
// See https://en.cppreference.com/w/c/string/byte/strlen

#include <stdio.h>

int StrLen1(const char *s);
int StrLen2(const char *s);
int StrLen3(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);
size_t StrNLen(const char *s, size_t max);

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

  printf("StrLen(%s) = %d\n", msg, StrLen1(msg));
  printf("StrLenStd(%s) = %zu\n", msg, StrLenStd(msg));

  return 0;
}

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

  return len;
}

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

  return len - 1;
}

int StrLen3(const char *s) {
  int len = -1;
  while (s[++len] != '\0');

  return len;
```

```c
}

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

  return sc - s;
}

// https://github.com/intel/safestringlib/blob/master/safeclib/strnlen_s.c
size_t StrNLen(const char *s, size_t max) {
  size_t count = 0;

  while (max && *s) {
    count++;
    max--;
    s++;
  }

  return count;
}
```

```
 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 2024/11/20.
```

```
1 add_executable(str-literals str-literals.c)
2 add_executable(strlen strlen.c)
3 add_executable(strcpy strcpy.c)
4 add_executable(strcmp strcmp.c)
5 add_executable(strcat strcat.c)
```

```c
//
// Created by hfwei on 2023/11/30.
// Visualization:
// https://pythontutor.com/render.html#code=%0A%23include%20%3Cstdio.h%3E%0A%23include%20%3Cstdlib.h%3E%0A%0Aint%20main%28void%29%20%7B%0A%20%20char%20msg%5B%5D%20%3D%20%22Hello%20World!%22%3B%0A%20%20msg%5B0%5D%20%3D%20'N'%3B%0A%20%20printf%28%22%25s%5Cn%22,%20msg%29%3B%0A%0A%20%20char%20*ptr_msg%20%3D%20%22Hello%20World!%22%3B%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&cumulative=true&curInstr=6&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>
#include <string.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
  // static storage duration (text segment)
  char *ptr_msg = "Hello Hello World!";
  char *ptr_msg_another = "Hello Hello World!";
  printf("%p\n%p\n", ptr_msg, ptr_msg_another);

  // undefined behavior

  // On Linux (or Debug mode on Windows): interrupted by signal 11: SIGSEGV
```

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

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

  return 0;
}
```

```c
// Benchmarking standard strncmp and STRNCMP in glibc.
// Created by hfwei on 2024/11/20.

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

// https://github.com/lattera/glibc/blob/master/string/strncmp.c
int STRNCMP(const char *s1, const char *s2, size_t n) {
  unsigned char c1 = '\0';
  unsigned char c2 = '\0';

  if (n >= 4) {
    size_t n4 = n >> 2;
    do {
      c1 = (unsigned char)*s1++;
      c2 = (unsigned char)*s2++;
      if (c1 == '\0' || c1 != c2)
        return c1 - c2;
      c1 = (unsigned char)*s1++;
      c2 = (unsigned char)*s2++;
      if (c1 == '\0' || c1 != c2)
        return c1 - c2;
      c1 = (unsigned char)*s1++;
      c2 = (unsigned char)*s2++;
      if (c1 == '\0' || c1 != c2)
        return c1 - c2;
      c1 = (unsigned char)*s1++;
      c2 = (unsigned char)*s2++;
      if (c1 == '\0' || c1 != c2)
        return c1 - c2;
    } while (--n4 > 0);
    n &= 3;
  }

  while (n > 0) {
    c1 = (unsigned char)*s1++;
    c2 = (unsigned char)*s2++;
    if (c1 == '\0' || c1 != c2)
      return c1 - c2;
    n--;
  }
```

```c

  return c1 - c2;
}

void benchmark(const char *label,
               int (*cmp_func)(const char *, const char *, size_t),
               const char *s1, const char *s2, size_t n, size_t iterations) {
  clock_t start = clock();

  for (size_t i = 0; i < iterations; i++) {
    cmp_func(s1, s2, n);
  }

  clock_t end = clock();
  double elapsed = (double)(end - start) / CLOCKS_PER_SEC;

  printf("%s: %.6f seconds\n", label, elapsed);
}

int main() {
  const size_t len = 1000000;
  const size_t iterations = 100000;
  char *s1 = malloc(len + 1);
  char *s2 = malloc(len + 1);

  memset(s1, 'a', len);
  memset(s2, 'a', len);
  s1[len] = '\0';
  s2[len] = '\0';

  printf("Benchmarking identical strings:\n");
  benchmark("STRNCMP", STRNCMP, s1, s2, len, iterations);
  benchmark("strncmp", strncmp, s1, s2, len, iterations);

  // Introduce a mismatch in the last character
  s2[len - 1] = 'b';

  printf("\nBenchmarking mismatched strings:\n");
  benchmark("STRNCMP", STRNCMP, s1, s2, len, iterations);
  benchmark("strncmp", strncmp, s1, s2, len, iterations);

  free(s1);
```

```
86    free(s2);
87
88    return 0;
89 }
```