A function to check for a string in a string

```
char* strstr(const char* haystack, const char* needle)
```

Implement your own `char* my_strstr(const char* haystack, const char* needle)`

This returns a pointer to where the string `needle` present in the string `haystack`

Couple of things to note:

- characters are integers at heart, int c is just the ASCII code for the character and can be tested for equality with characters
- the functions should handles nulls and return a null if the needle is not found in the haystack

SUBSTRING CHECK IMPLEMENTATION

```
char* my_strstr(const char* haystack, const char* needle) {
  if (haystack == NULL || needle == NULL) {
    return NULL;
  }

  while (*haystack != '\0') {
    if (*haystack == *needle) {
      const char* h = haystack;
      const char* n = needle;
      while (*n != '\0' && *h == *n) {
        h++;
        n++;
      }
      if (*n == '\0') {
        return (char*) haystack;
      }
    }
    haystack++;
  }

  return NULL;
}
```

THE FUNCTION SHOULD NOT FAIL ON NULL INPUTS!

IF WE FIND A MATCH WITH THE STARTING CHARACTER OF NEEDLE WE CHECK THE ENTIRE NEEDLE STRING FROM THAT POINT ONWARDS

INITIALIZE NEW POINTERS TO WALK THE HAYSTACK AND NEEDLE STRINGS IN TANDEM

CONTINUE CHECKING THE INDIVIDUAL CHARACTERS AS LONG AS THEY ARE EQUAL

IF WE'VE REACHED THE END OF THE NEEDLE STRING IT MEANS THAT WE'VE FOUND AN INSTANCE OF NEEDLE IN THE HAYSTACK

IF WE COME HERE THEN, WE HAVEN'T FOUND A MATCH BETWEEN NEEDLE AND A SUBSTRING IN HAYSTACK, WE MOVE ON TO START COMPARING THE NEXT CHARACTER IN HAYSTACK

A function to compare two strings

```
int strcmp(const char* str1, const char* str2)
```

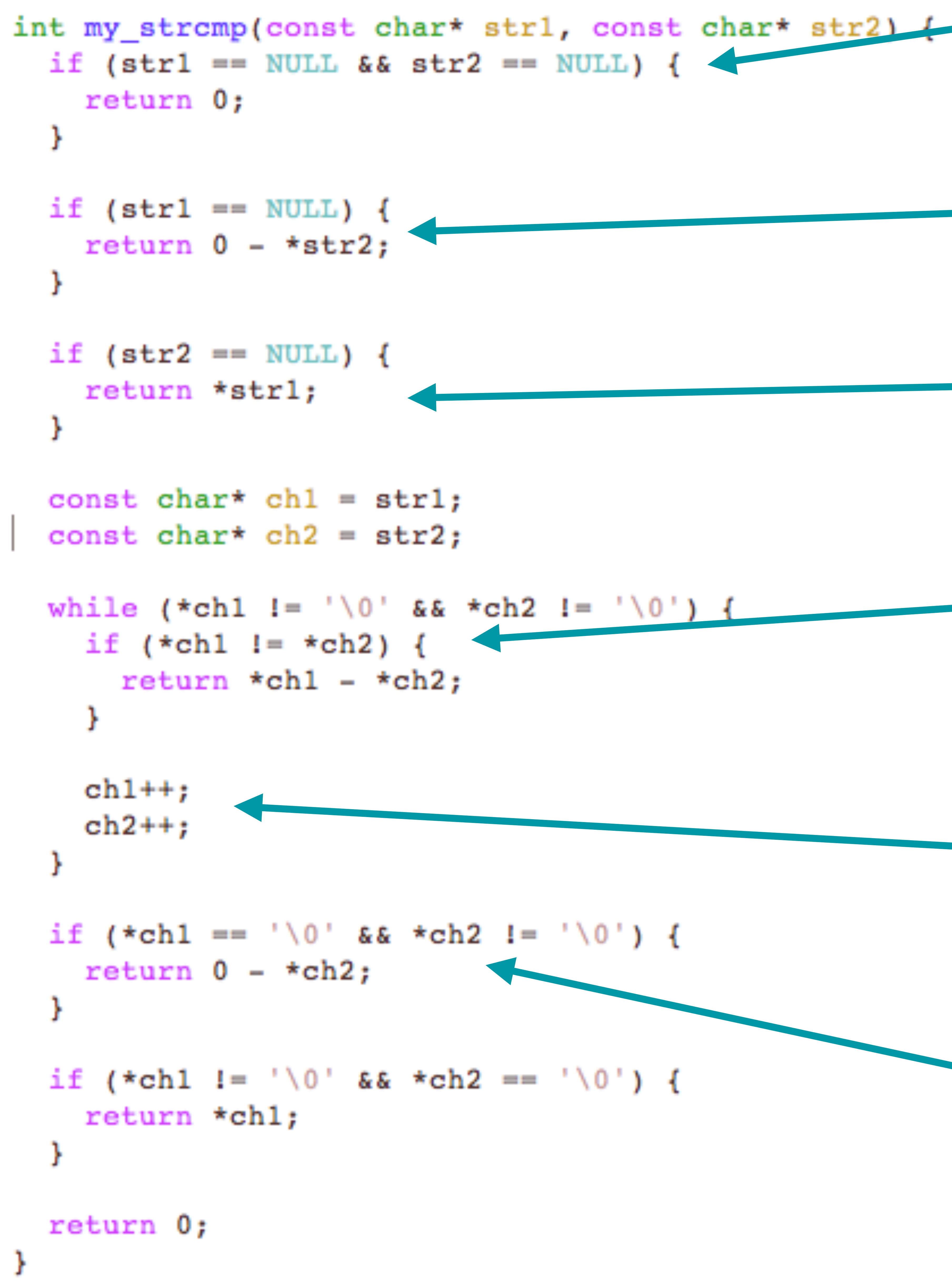
Implement your own `int my_strcmp(const char* str1, const char* str2)`

This returns a number
 < 0 if str1 is less than str2,
 > 0 if str1 is greater than str2
 == 0 if str1 is equal to str2

Couple of things to note:

- It should be able to handle null inputs
- Makes sure that it works for all string lengths and handles errors correctly
- ASCII characters are integers at heart, they correspond to ASCII codes 'A' starts at 65 and 'a' starts at 97 with the other capital and smaller case letters following them.

STRING LENGTH IMPLEMENTATION



```c
int my_strcmp(const char* str1, const char* str2) {
  if (str1 == NULL && str2 == NULL) {
    return 0;
  }

  if (str1 == NULL) {
    return 0 - *str2;
  }

  if (str2 == NULL) {
    return *str1;
  }

  const char* ch1 = str1;
  const char* ch2 = str2;

  while (*ch1 != '\0' && *ch2 != '\0') {
    if (*ch1 != *ch2) {
      return *ch1 - *ch2;
    }

    ch1++;
    ch2++;
  }

  if (*ch1 == '\0' && *ch2 != '\0') {
    return 0 - *ch2;
  }

  if (*ch1 != '\0' && *ch2 == '\0') {
    return *ch1;
  }

  return 0;
}
```

THE FUNCTION SHOULD NOT FAIL
ON NULL INPUTS!

IF STR1 IS NULL THEN CONSIDER
IT SMALLER THAN STR2

IF STR2 IS NULL THEN CONSIDER
IT SMALLER THAN STR1

IF THERE IS A MISMATCH WE SIMPLY SUBTRACT THE
ASCII CODES OF THE LETTER AT WHICH THE MISMATCH
OCCURS. THIS DIRECTLY TELLS US WHETHER THE
ALPHABET AT THE SPECIFIED INDEX IN STR1 IS LESS
THAN OR GREATER THAN THE CORRESPONDING
ALPHABET IN STR2

INCREMENT BOTH POINTERS TO GET AT THE NEXT CHARACTER,
WE COME HERE ONLY IF THE CHARACTERS SEEN SO FAR IN
BOTH STRINGS ARE EQUAL

IF THE LENGTHS OF THE STRINGS ARE UNEQUAL ONE OF
THEM WILL RUN OUT FIRST, THE STRING OF THE
SMALLER LENGTH IS CONSIDERED LESS THAN THE OTHER

A function to concatenate strings

```
char* strcat(char* dest, char* src)
```

Implement your own `char* my_strcat(char* dest, char* src)`

This returns a pointer to the destination string which has the complete concatenated value.

Couple of things to note:

- copying can be done character by character
- assume there is enough space in the destination string to hold the source appended to it

STRING CONCATENATION IMPLEMENTATION

```
char* my_strcat(char* dest, const char* src) {
  if (dest == NULL || src == NULL) {
    return dest;
  }

  char* d = dest;
  while (*d != '\0') {
    d++;
  }

  while (*src != '\0') {
    *d = *src;
    d++;
    src++;
  }
  *d = '\0';

  return dest;
}
```

IF ANY OF THE INPUT STRINGS ARE NULL WE RETURN A POINTER TO THE DESTINATION WHICH WILL BE NULL

USE A VARIABLE TO MOVE OVER THE DESTINATION STRING CHARACTER BY CHARACTER, SO THAT THE ORIGINAL DEST POINTER IS NOT DISTURBED

ASSUME THAT THE DESTINATION STRING HAS SUFFICIENT MEMORY TO HOLD THE SOURCE AS WELL