APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 21
March 9, 2012

Today

- Even More Strings

- Midterms

Midterm

- Average was 71.3%

Question 3. [12 MARKS]

Here is the documentation and prototype for a function called `calculateSumOfSquares()`:

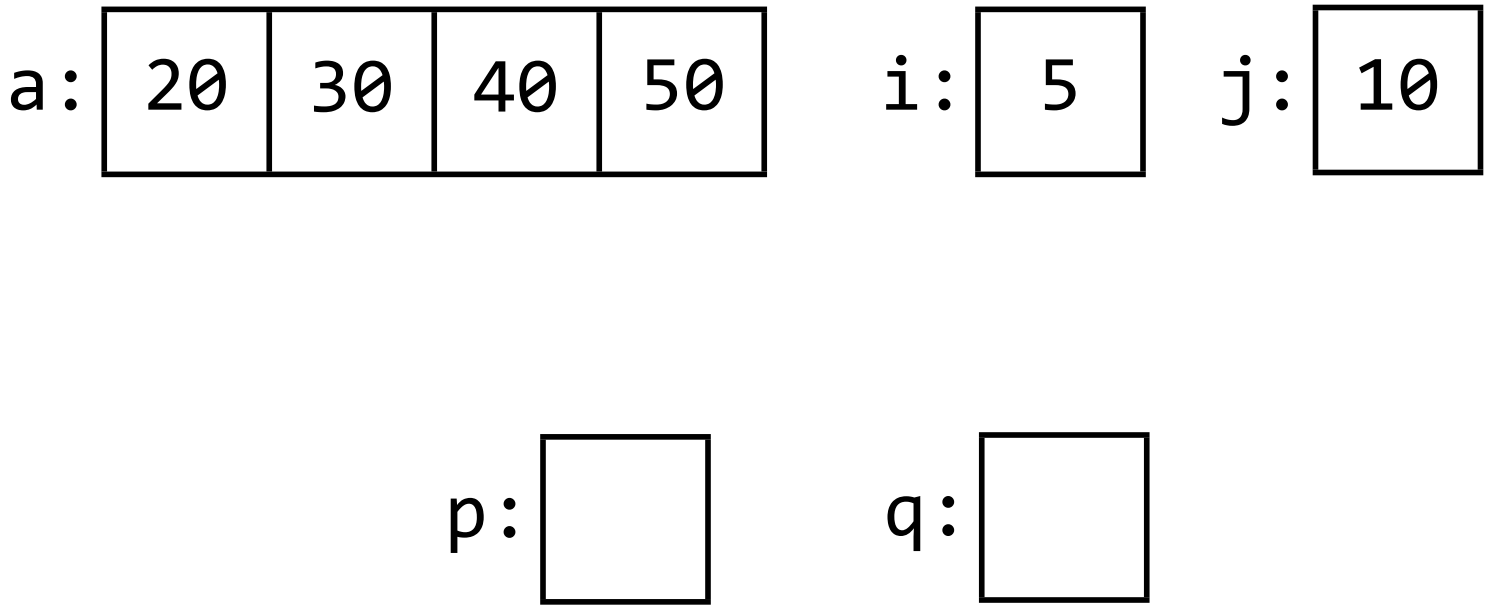
```
/* Returns the sum of the square of each number in 'list'.
 * 'list' is of length 'n'.  If 'list' is of length 0, returns -1. */
int calculateSumOfSquares(const int list[], int n);
```

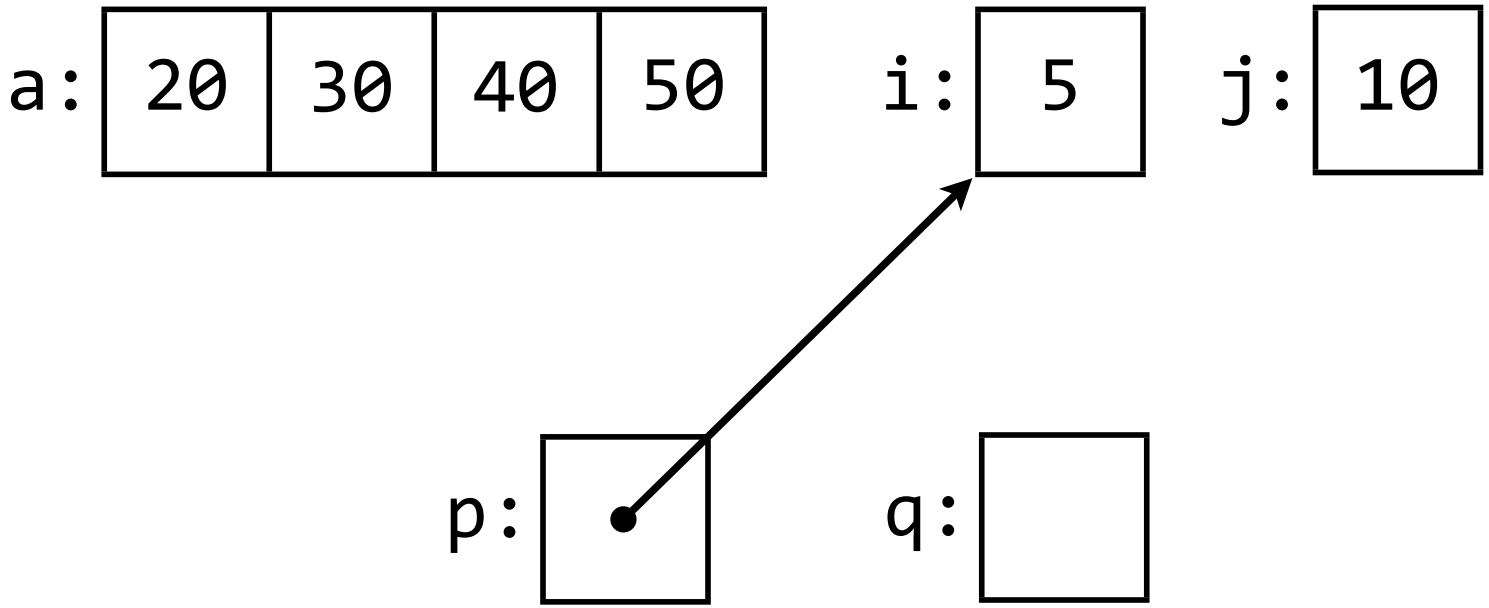
Part (a) [2 MARKS]

Assume you have an `int[]` of length `n` called `observations`. Write a code snippet that prints out the sum of the squares of the numbers in `observations`, or `"Empty dataset"` if `observations` is empty. Your snippet should call `calculateSumOfSquares()`.

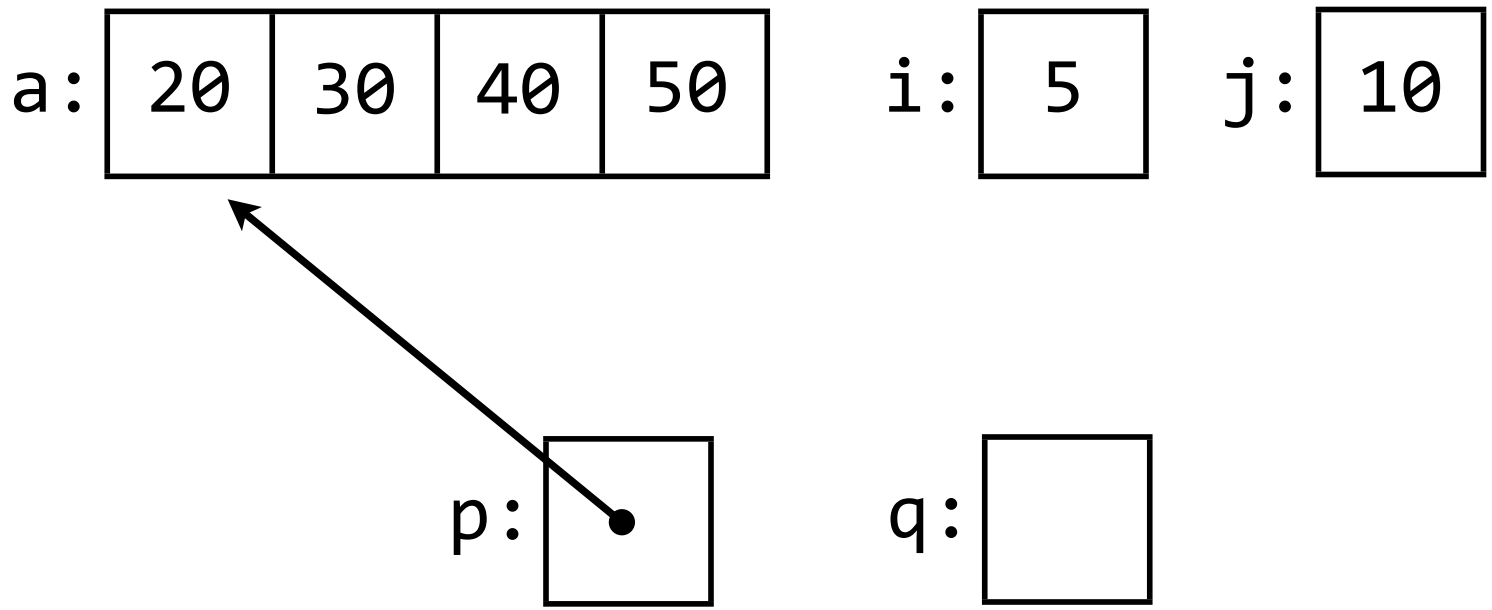
```
int sumOfSquares = calculateSumOfSquares(occurrences, n);
if (sumOfSquares == -1)
{
   printf("Empty dataset\n");
}
else
{
   printf("%d\n", sumOfSquares);
}
```

```
-----
-----
-----
-----
-----
Enter a row: 2
-----
-----
*****
-----
-----
```

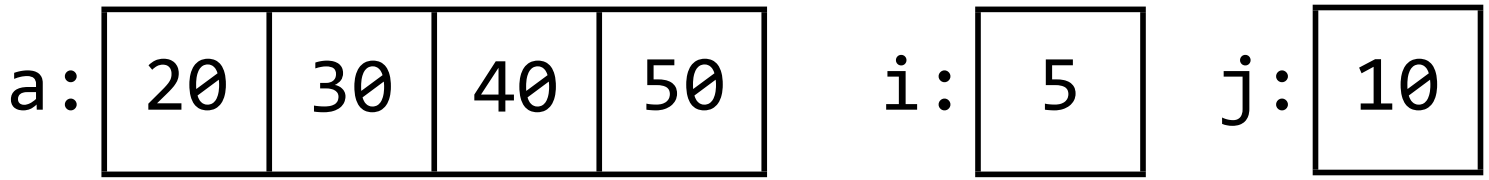

a:
20
30
40
50
i:
5
j:
10
p:
q:




```
p = &i;
```

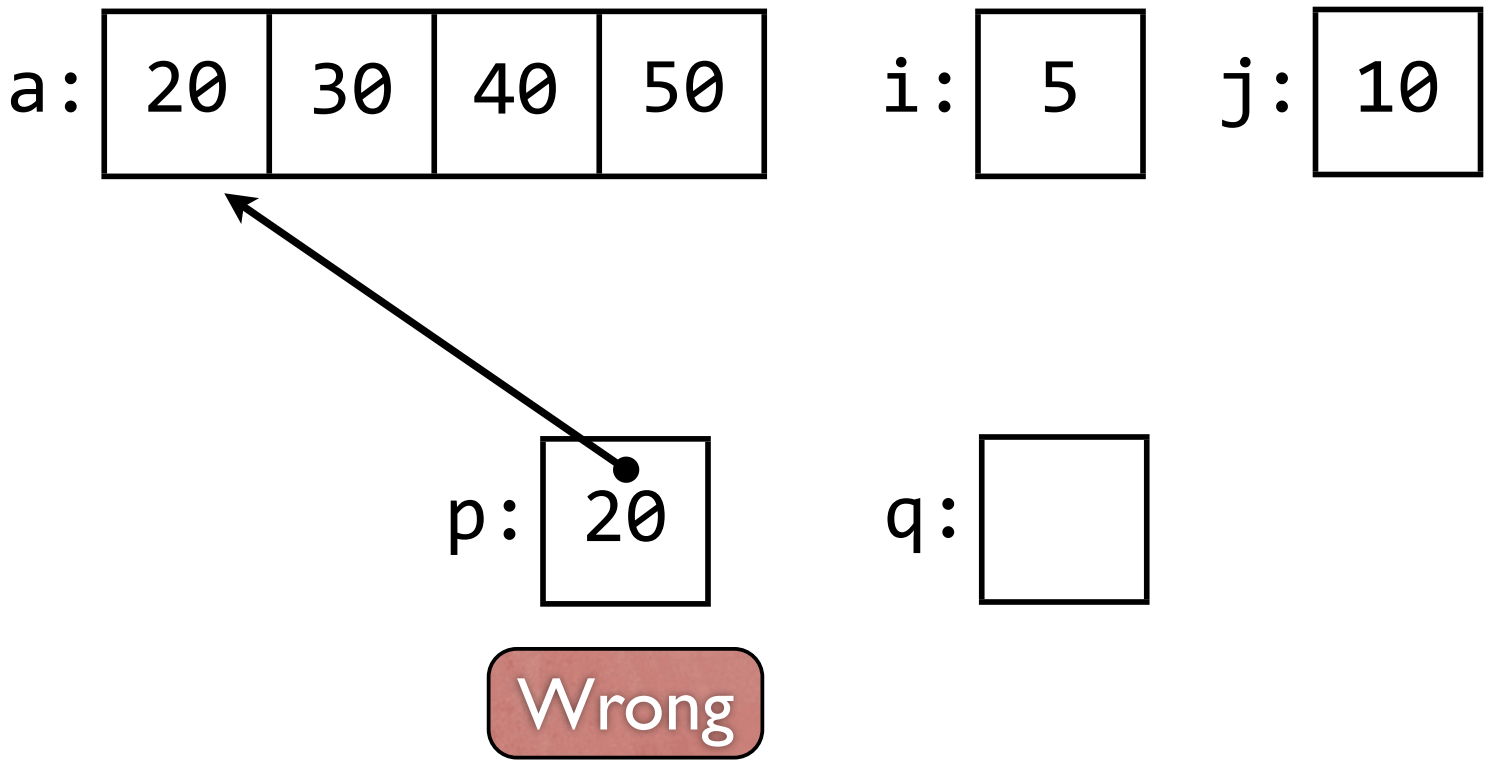



```
p = &i;
p = a;
```



Wrong

```
p = &i;
p = a;
```




```
p = &i;
p = a;
```

Strings
(recap)

`string.h`

- The C string library
- `#include <string.h>` (no `-l` needed)
- Provides a bunch of useful string operations
- Works around some C limitations

strlen()

for now think of it as int

```
size_t strlen(const char *s);
```

- Returns the length of s
- The number of characters up to but *not including* '\0'

```
#define MAX_LEN 50
...
char str1[MAX_LEN + 1] = "Hello";
int length = strlen(str1);    5
length = strlen("");          0
```

```
char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

str1 = "Hello";
```
Error

```
str1 = {'H','e','l','l','o','\0'};
```
Error

```
int a[4];
a = {2, 4, 6, 8};
```
Error

```
char str3[MAX_LEN + 1] = {'H','i', '\0'};


str1 = str3;
```
Error

strcpy()

```
char *strcpy(char *s1, const char *s2);
```

- Copies the string in s2 into s1

```
char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1] = "Hello";

str1 = str2;
```

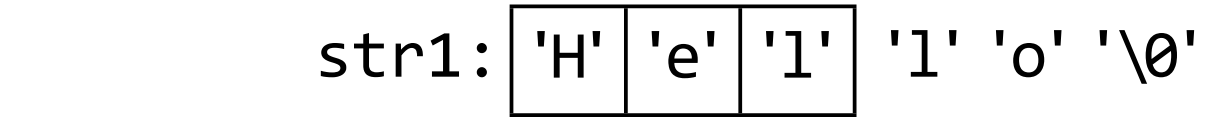
Error

```
strcpy(str1, str2);
```

"Copies the string in `s2` into `s1`"

Formally, copies every character in `s2` into `s1`, up to and including the first `\0`

```
char str1[2 + 1];
char str2[MAX_LEN + 1] = "Hello";

strcpy(str1, str2);
```



Never Use `strcpy()`!

- `strcpy()` is inherently unsafe

You are not allowed to use `strcpy()` on assignments and tests

strncpy()

```
char *strncpy(char *s1, const char *s2, int n);
```

- Safer, but not completely safe
- Copies at most `n` characters from the string in `s2` into `s1`

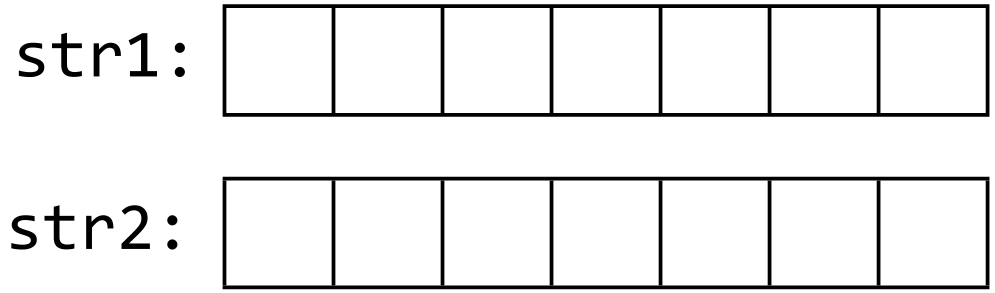
```
char str1[2 + 1];
char str2[MAX_LEN + 1] = "Hello";

strcpy(str1, str2);
```

Wrong

```
strncpy(str1, str2, 2 + 1);
```

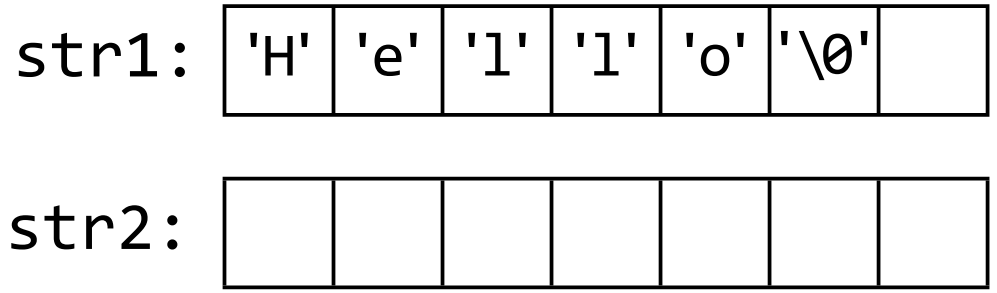
Why Only "Safer"?

```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];
```



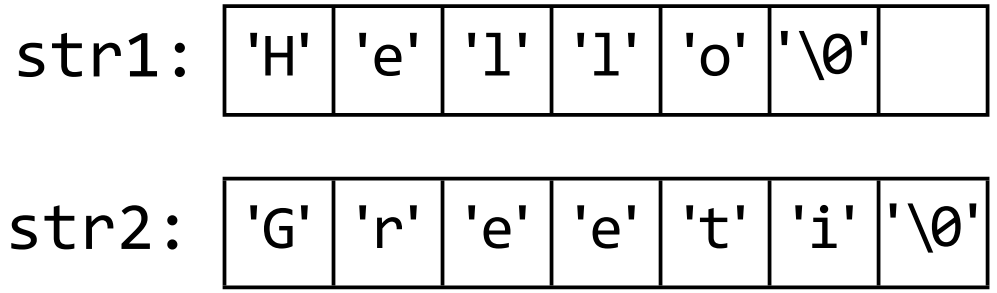
Why Only "Safer"?

```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

strncpy(str1, "Hello", MAX_LEN + 1);
```



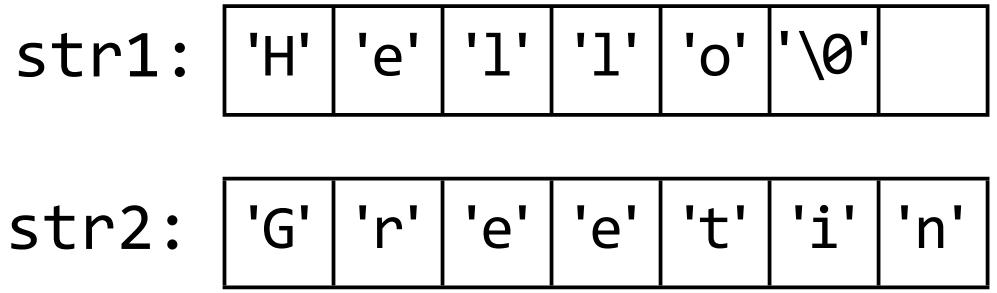
Why Only "Safer"?

```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

strncpy(str1, "Hello", MAX_LEN + 1);

strncpy(str2, "Greetings", MAX_LEN + 1);
```



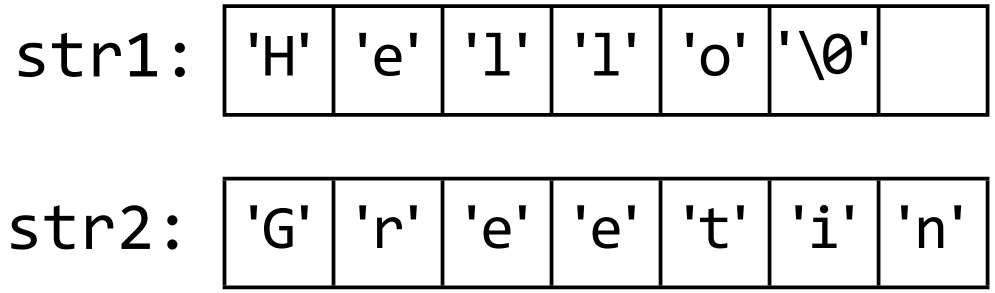
Why Only "Safer"?

```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

strncpy(str1, "Hello", MAX_LEN + 1);

strncpy(str2, "Greetings", MAX_LEN + 1);
```



Why Only "Safer"?

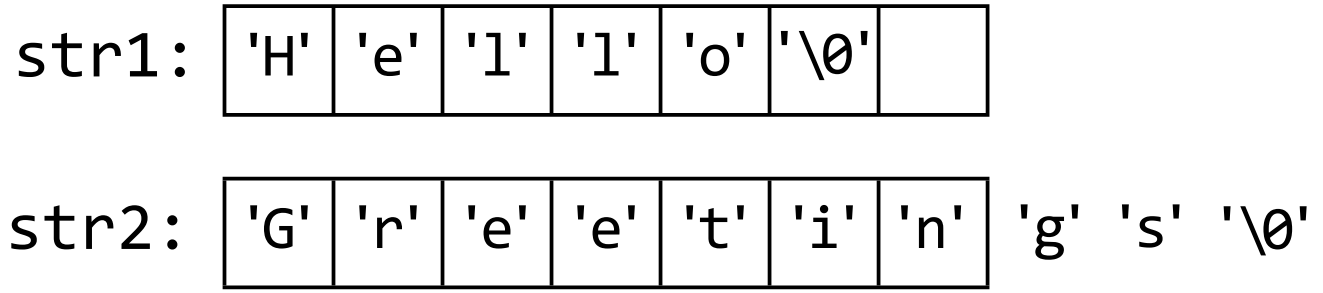
```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

strncpy(str1, "Hello", MAX_LEN + 1);

strncpy(str2, "Greetings", MAX_LEN + 1);
```



str2 is no longer a string

Why Only "Safer"?

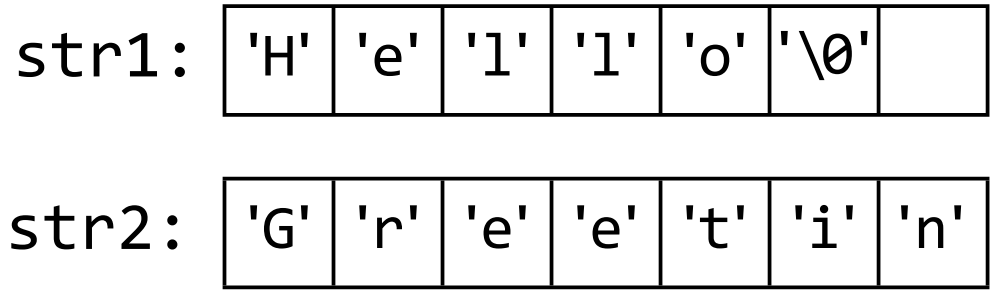
```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

strncpy(str1, "Hello", MAX_LEN + 1);

strncpy(str2, "Greetings", MAX_LEN + 1);
```



str2 is no longer a string

Why Only "Safer"?

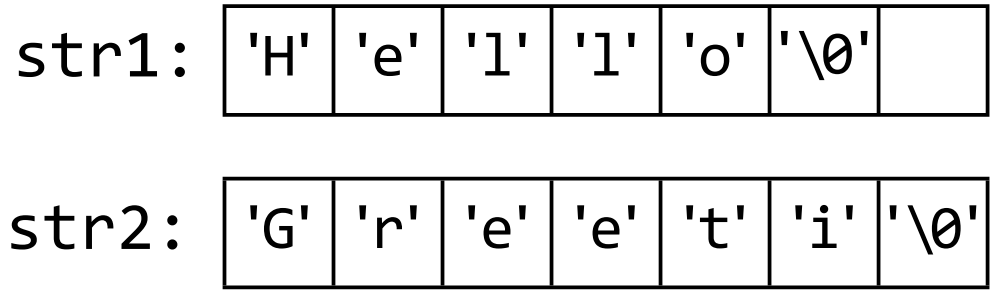
```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

strncpy(str1, "Hello", MAX_LEN + 1);

strncpy(str2, "Greetings", MAX_LEN + 1);
```



str2 is no longer a string

Why Only "Safer"?

```
#define MAX_LEN 6

char str1[MAX_LEN + 1];
char str2[MAX_LEN + 1];

strncpy(str1, "Hello", MAX_LEN + 1);

strncpy(str2, "Greetings", MAX_LEN + 1);

str2[MAX_LEN] = '\0';
```



str2 is no longer a string

Strings
(not a recap)

strcat()

```
char *strcat(char *s1, const char *s2);
```

- Copies the string in s2 onto the end of s1
- "Cat" is concatenation

```
char str1[MAX_LEN + 1] = "Hello ";
char str2[MAX_LEN + 1] = "World";

str1 = str1 + str2;
```

Error

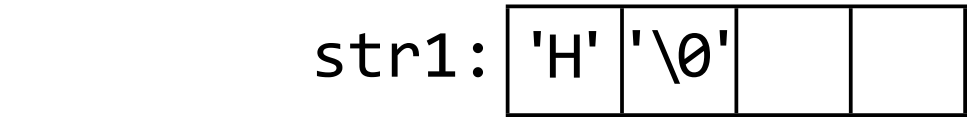
```
strcat(str1, str2);
printf("%s", str1);
```

```
Hello World
```

"Copies the string in `s2` on to the end of `s1`"

Formally, finds the first `\0` in `s1`. Starting at that location, copies every character in `s2` into `s1`, up to and including the first `\0`

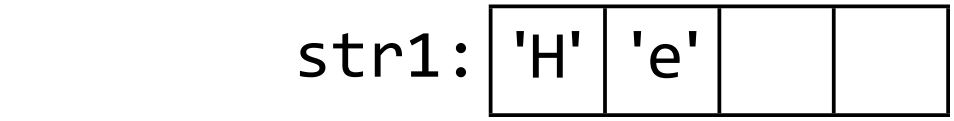
```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strcat(str1, str2);
```



"Copies the string in `s2` on to the end of `s1`"

Formally, finds the first `\0` in `s1`. Starting at that location, copies every character in `s2` into `s1`, up to and including the first `\0`

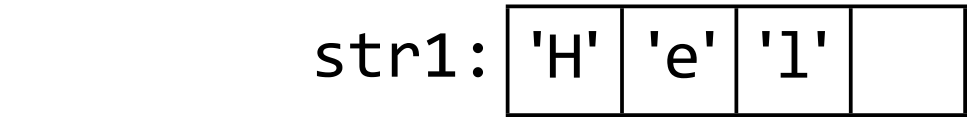
```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strcat(str1, str2);
```



"Copies the string in `s2` on to the end of `s1`"

Formally, finds the first `\0` in `s1`. Starting at that location, copies every character in `s2` into `s1`, up to and including the first `\0`

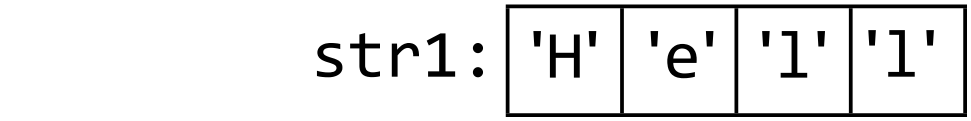
```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strcat(str1, str2);
```



"Copies the string in `s2` on to the end of `s1`"

Formally, finds the first `\0` in `s1`. Starting at that location, copies every character in `s2` into `s1`, up to and including the first `\0`

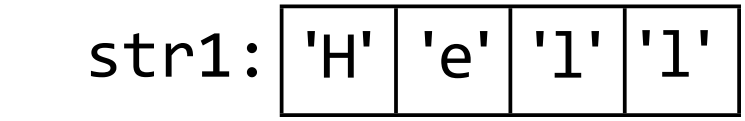
```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strcat(str1, str2);
```



"Copies the string in `s2` on to the end of `s1`"

Formally, finds the first `\0` in `s1`. Starting at that location, copies every character in `s2` into `s1`, up to and including the first `\0`

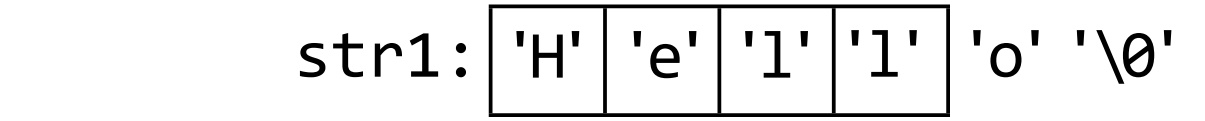
```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strcat(str1, str2);
```



"Copies the string in `s2` on to the end of `s1`"

Formally, finds the first `\0` in `s1`. Starting at that location, copies every character in `s2` into `s1`, up to and including the first `\0`

```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strcat(str1, str2);
```



Never Use `strcat()`!

- `strcat()` is inherently unsafe

You are not allowed to use `strcat()` on assignments and tests

strncat()

```
char *strncat(char *s1, const char *s2, int n);
```

- Unlike `strncpy()`, `strncat()` is completely safe
- Copies at most `n` characters from the string in `s2` into `s1`, then appends a `\0`

```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strcat(str1, str2);
```

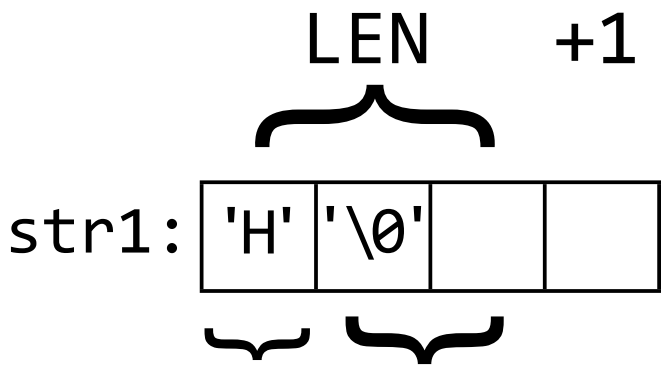
Wrong

```
strncat(str1, str2, ?);
```

Incorrect n is a Common Bug

Copies at most n characters from the string in s2 into s1, then appends a \0

```
#define LEN 3
char str1[LEN + 1] = "H";
```



n = total size of array - 1 - length of existing string

```
n = (LEN + 1) - 1 - strlen(str1)
n = LEN - strlen(str1)
```

strncat()

```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strncat(str1, str2, ? );
```

strncat()

n is LEN - strlen(str1)

```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strncat(str1, str2, 3 - strlen(str1) );
```

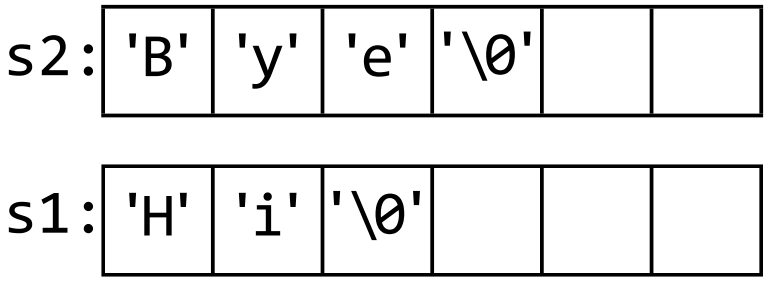
strncat()

n is LEN - strlen(str1)

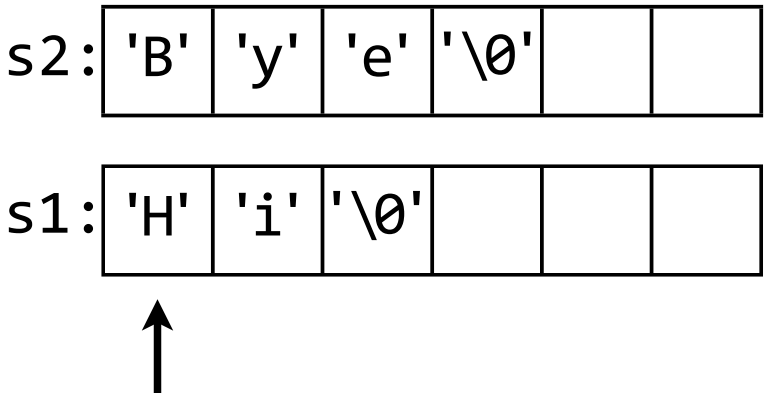
```
char str1[3 + 1] = "H";
char str2[MAX_LEN + 1] = "ello";

strncat(str1, str2, 3 - strlen(str1) );

strncat(str1, str2, 3);
strncat(str1, str2, 3 + 1);
```

Wrong

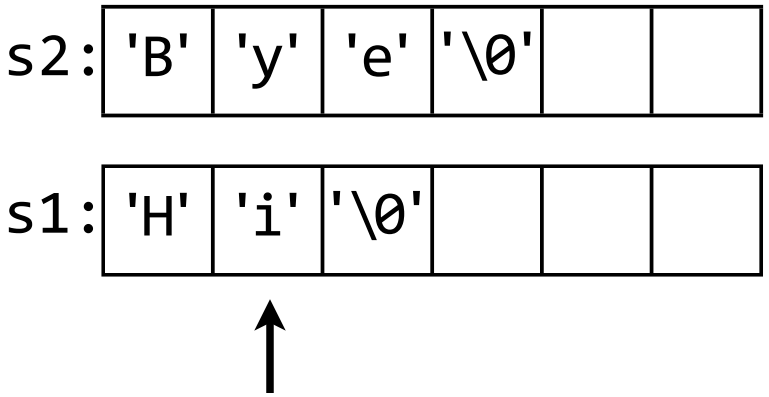
Wrong



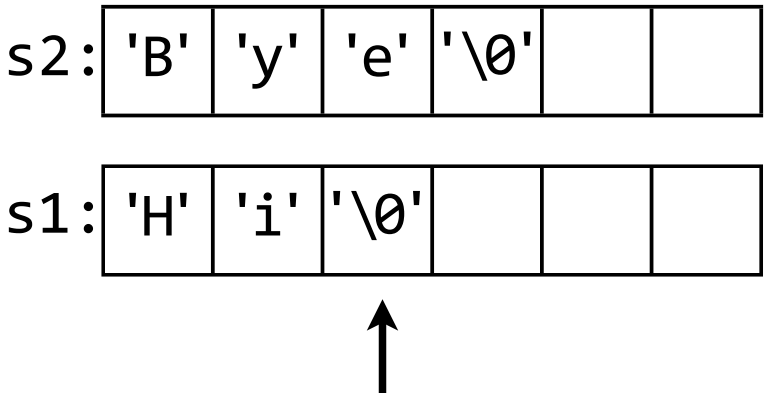
Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



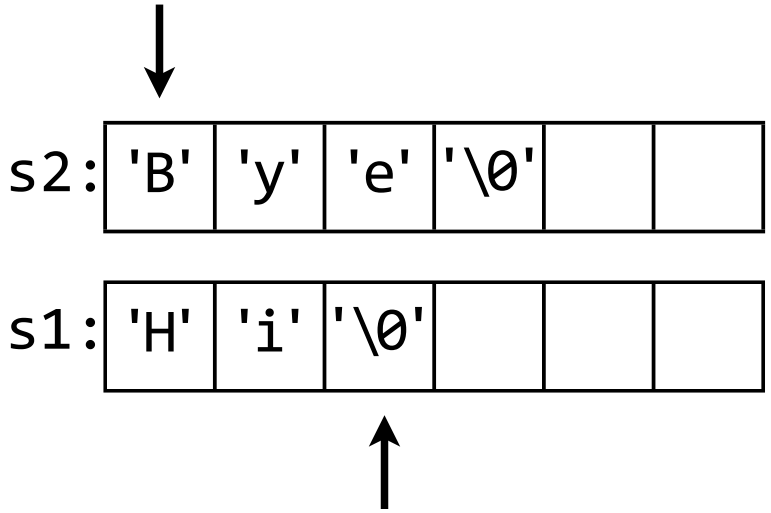
Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



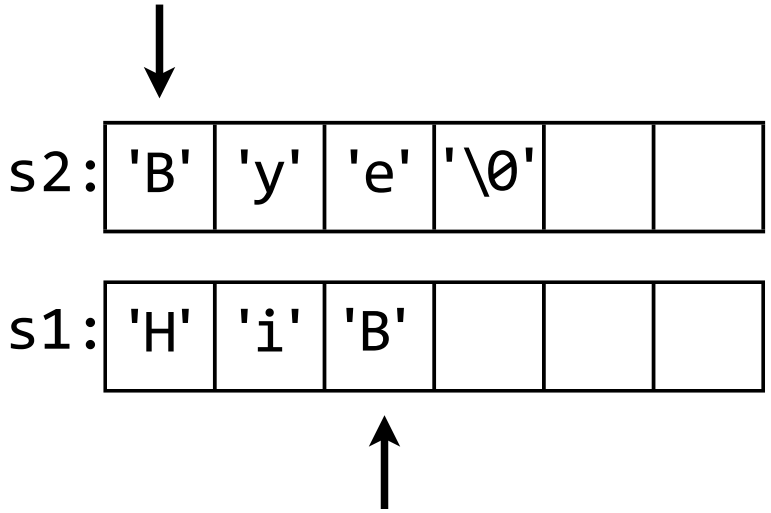
Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



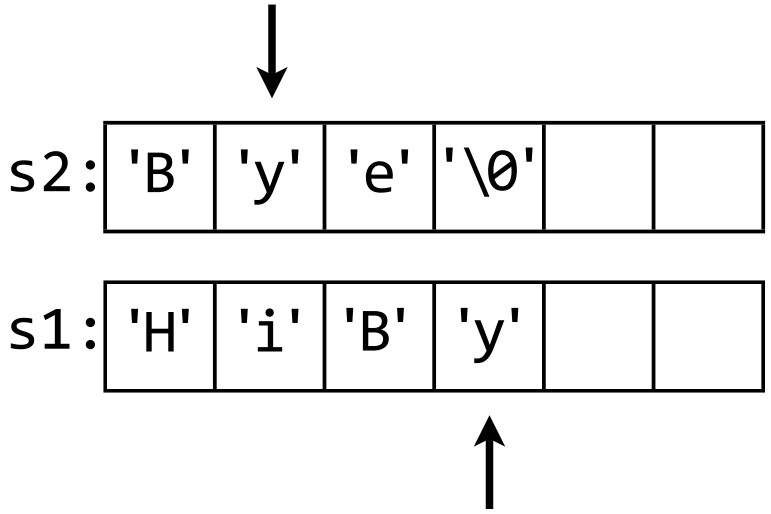
Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



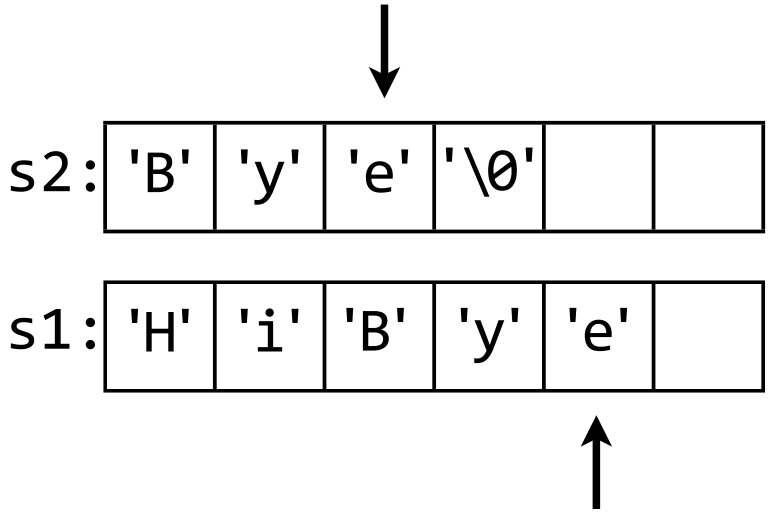
Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



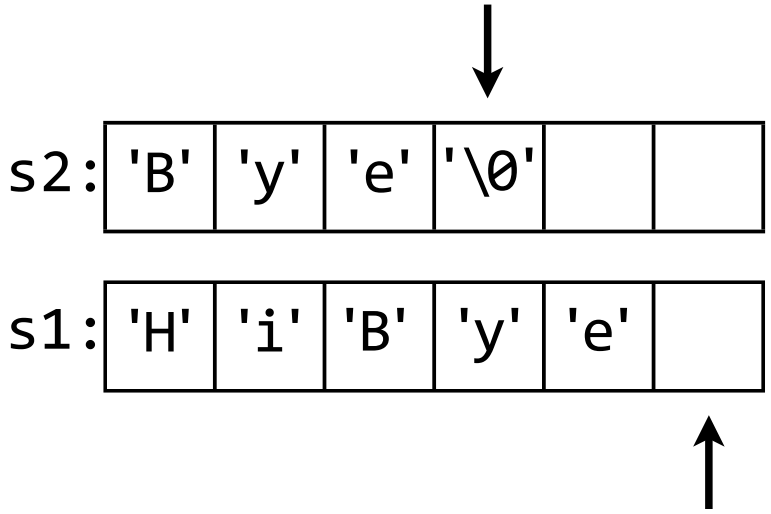
Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



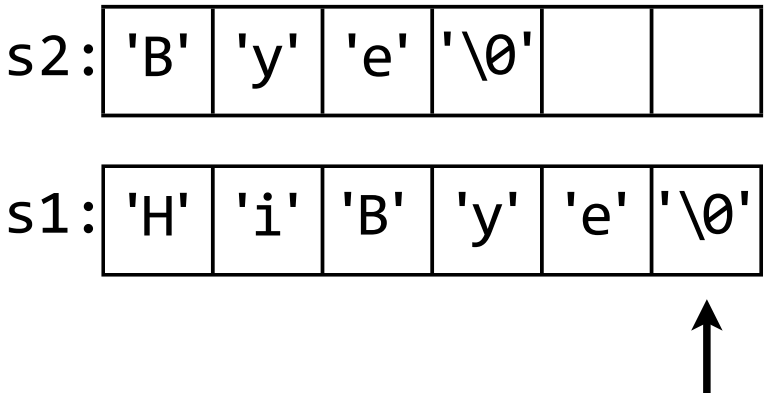
Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`



Find the first `\0` of `s1`
Start at the first `char` of `s2`
`while` (the current `char` is not `\0`)
{
  Copy that character to `s1`
  Move on to the next `char`
}
Add a `\0` at the current position in `s1`

Find the first \0 of s1
Start at the first char of s2
while (the current char is not \0)
{
  Copy that character to s1
  Move on to the next char
}
Add a \0 at the current position in s1

```
void my_strcat(char *s1, const char *s2)
{
    int s1Length = 0;

    while(s1[s1Length] != '\0')
    {
        s1Length++;
    }

    int i;
    for (i = 0; s2[i] != '\0'; i++)
    {
        s1[s1Length + i] = s2[i];
    }
    s1[s1Length + i] = '\0';
}
```

Find the first \0 of s1
Start at the first char of s2
while (the current char is not \0)
{
  Copy that character to s1
  Move on to the next char
}
Add a \0 at the current position in s1

```
void my_strcat(char *s1, const char *s2)
{
    int s1Length = my_strlen(s1);
    int i;
    for (i = 0; s2[i] != '\0'; i++)
    {
        s1[s1Length + i] = s2[i];
    }
    s1[s1Length + i] = '\0';
}
```

```
void my_strcat(char *s1, const char *s2)
{
    int s1Length = my_strlen(s1);
    int i;
    for (i = 0; s2[i] != '\0'; i++)
    {
        s1[s1Length + i] = s2[i];
    }
    s1[s1Length + i] = '\0';
}
```

```
void my_strncat(char *s1, const char *s2, int n)
{
    int s1Length = my_strlen(s1);
    int i;
    for (i = 0; s2[i] != '\0'; i++)
    {
        s1[s1Length + i] = s2[i];
    }
    s1[s1Length + i] = '\0';
}
```

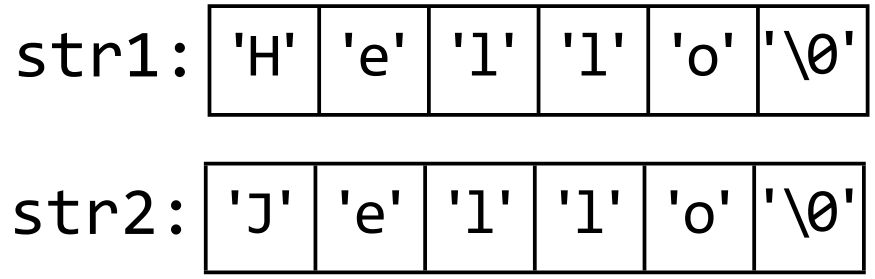
```
void my_strncat(char *s1, const char *s2, int n)
{
    int s1Length = my_strlen(s1);
    int i;
    for (i = 0; s2[i] != '\0' && i < n; i++)
    {
        s1[s1Length + i] = s2[i];
    }
    s1[s1Length + i] = '\0';
}
```

strcmp()

```
int strcmp(const char *s1, const char *s2);
```

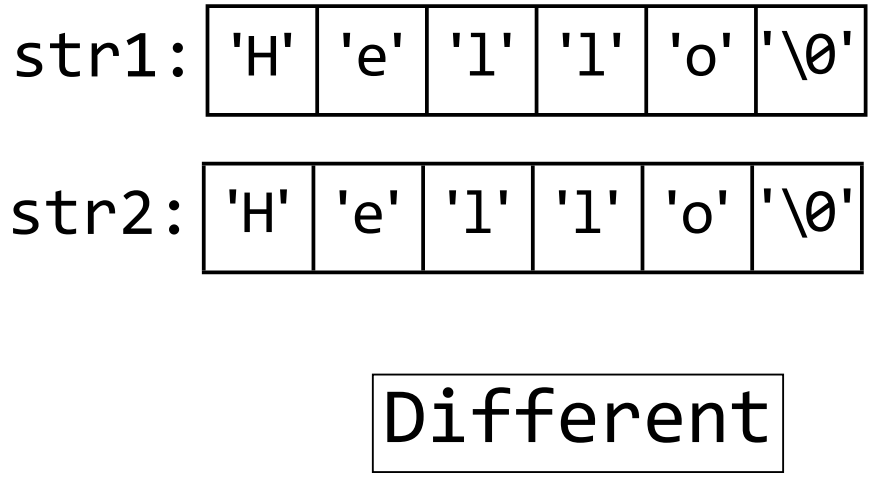
- Compares `s1` and `s2`

```
char str1[MAX_LEN + 1] = "Hello";
char str2[MAX_LEN + 1] = "Jello";

if (str1 == str2)
{
   printf("Same");
}
else
{
   printf("Different");
}
```

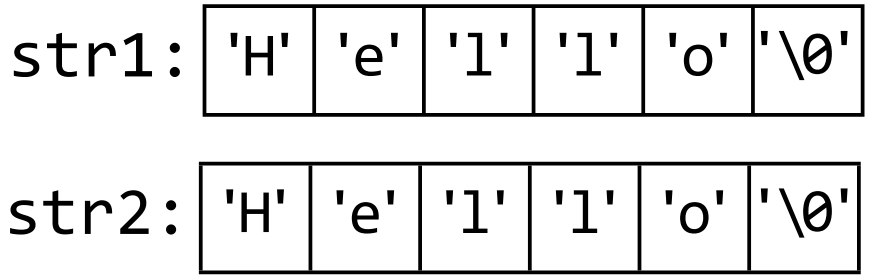


Different

```
char str1[MAX_LEN + 1] = "Hello";
char str2[MAX_LEN + 1] = "Jello";

if (str1 == str2)
{
   printf("Same");
}
else
{
   printf("Different");
}
str2[0] = 'H';
```



```
char str1[MAX_LEN + 1] = "Hello";
char str2[MAX_LEN + 1] = "Jello";

if (str1 == str2)
{
   printf("Same");
}
else
{
   printf("Different");
}
str2[0] = 'H';
if (str1 == str2)
{
   printf("Same");
}
else
{
   printf("Different");
}
```



Different

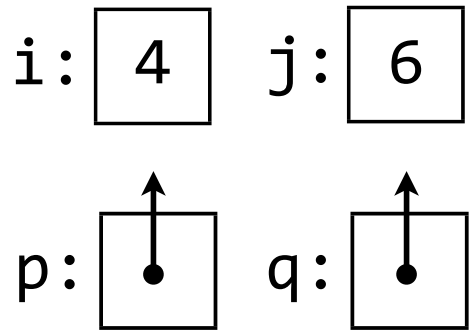
Different

Equality

- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

`false`



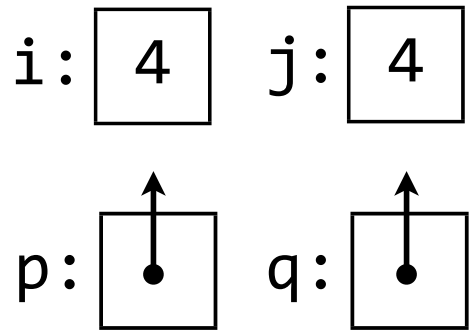
Equality

- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

`false`

```
j = 4;
```



Equality

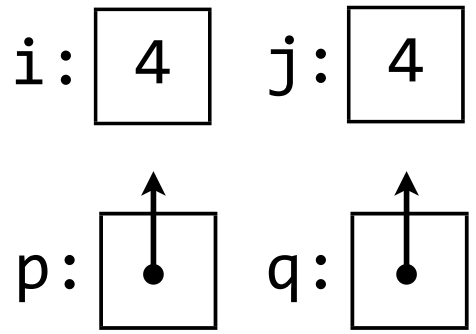
- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

`false`

```
j = 4;
b = (p == q);
```

`false`



Equality

- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

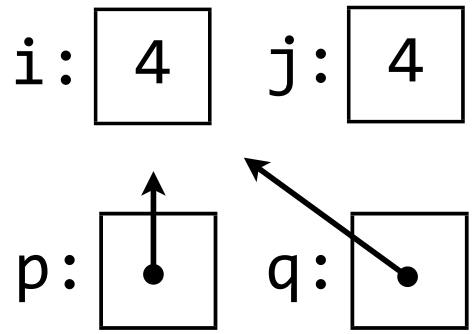
```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

`false`

```
j = 4;
b = (p == q);
```

`false`

```
q = &i;
```



Equality

- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

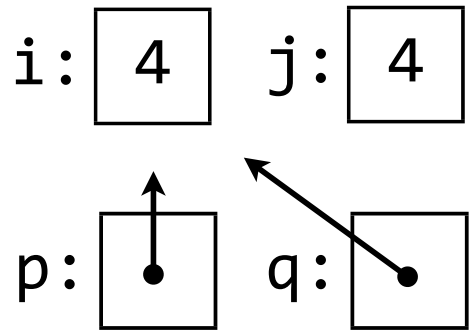
false

```
j = 4;
b = (p == q);
```

false

```
q = &i;
b = (p == q);
```

true



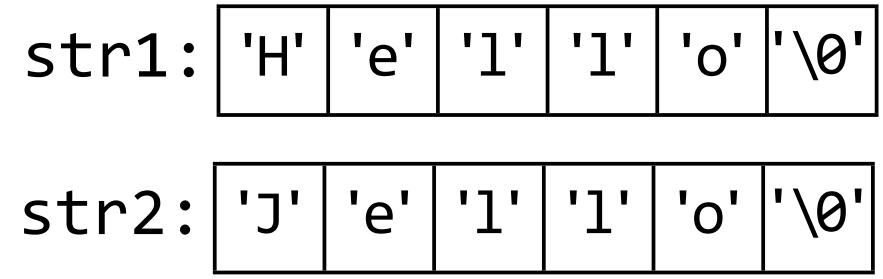
strcmp()

```
int strcmp(const char *s1, const char *s2);
```

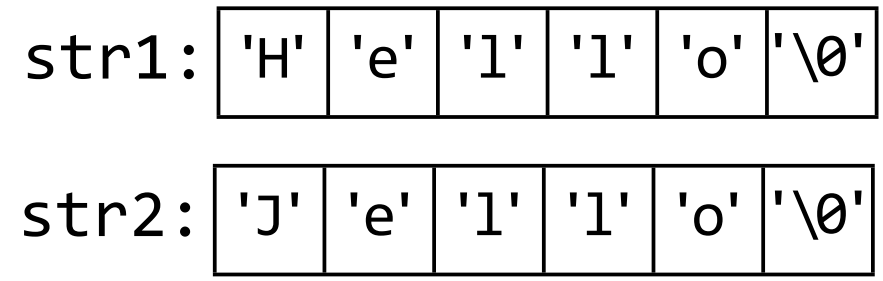
- Compares s1 and s2
  - Returns 0 if s1 and s2 are equal
  - Returns < 0 if s1 is before s2
  - Returns > 0 if s1 is after s2
- Use relational/equality operators to test against 0

is s1 after s2?	`strcmp(s1, s2) > 0`
is s1 equal to s2?	`strcmp(s1, s2) == 0`

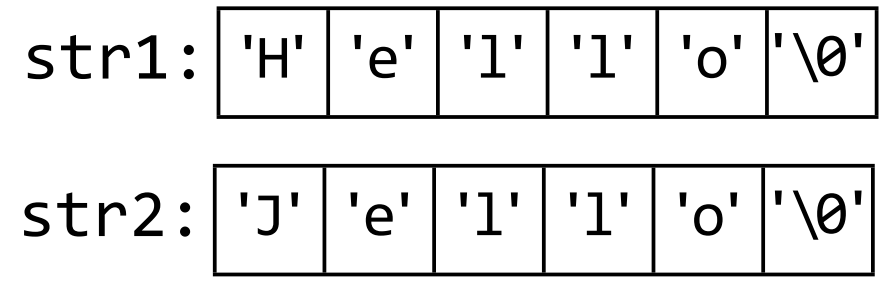
```
char str1[MAX_LEN + 1] = "Hello";
char str2[MAX_LEN + 1] = "Jello";

if (str1 == str2)
{
   printf("Same");
}
else
{
   printf("Different");
}
str2[0] = 'H';
if (str1 == str2)
{
   printf("Same");
}
else
{
   printf("Different");
}
```



```
char str1[MAX_LEN + 1] = "Hello";
char str2[MAX_LEN + 1] = "Jello";

if (strcmp(str1, str2) == 0)
{
   printf("Same");
}
else
{
   printf("Different");
}
str2[0] = 'H';
if (strcmp(str1, str2) == 0)
{
   printf("Same");
}
else
{
   printf("Different");
}
```



```
char str1[MAX_LEN + 1] = "Hello";
char str2[MAX_LEN + 1] = "Jello";

if (strcmp(str1, str2) == 0)
{
   printf("Same");
}
else
{
   printf("Different");
}
str2[0] = 'H';
if (strcmp(str1, str2) == 0)
{
   printf("Same");
}
else
{
   printf("Different");
}
```



Different

Same

How Does It Compare?

- Lexicographical
- "Dictionary order"

```
"abcd" < "fghi"
"abc"  < "abcdef"
```

- Uses ASCII ("ASCIIbetical")

space < digits < UPPER CASE < lower case

```
"ABC" < "DEF"
"Zoo" < "apple"
" z"  < "123"
```

start with the first char of s1 and s2
while (they are the same)
{
  if (they are \0)
  {
    return 0
  }
  move on to the next characters
}
return (current char of s1) - (current char of s2)

```
int my_strcmp(const char *s1, const char *s2)
{
    int i;
    for (i = 0; s1[i] == s2[i]; i++)
    {
        if (s1[i] == '\0')
        {
            return 0;
        }
    }
    return s1[i] - s2[i];
}
```