APS105
Winter 2012

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

Lecture 20
March 7, 2012

Today

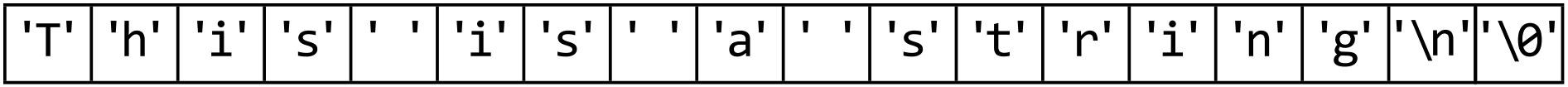
- More Strings

- No CodeLab this week

Strings
(recap)

```
char s[] = "This is a string\n";

char s[] = {'T','h','i','s',' ','i','s',' ','a',
            ' ','s','t','r','i','n','g','\n','\0'};
```

s:



C String Issues

- Strings don't know how long they are
  - Need to search for the `'\0'`
- Arrays don't know how big they are
  - Don't know if a string will fit into a given `char[]`

Printing Strings

- `printf()` has a `%s` conversion specifier

```
char message1[] = "Hello everyone.\n";
printf(message1);
```

Hello everyone.

```
char message2[] = "50% Off!\n";
printf(message2);
```

Wrong

??

```
printf("%s", message2);
```

50% Off!

```
char trouble[2] = "Hi";
printf("%s", trouble);
```

??

Wrong

Never Use `gets()`!

- `gets()` is inherently unsafe

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

fgets()

```
char *fgets(char *s, int size, FILE *stream);
```

- Reads a single line of at most `size - 1` characters, keeps the `'\n'` (if room), adds `'\0'`
- The stream is where to read from, for us it's always `stdin`

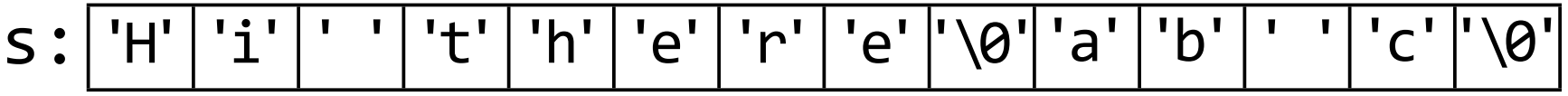
```
char *fgets(char *s, int size, FILE *stream);


#define INPUT_LEN 5
...

char input[INPUT_LEN + 1];
fgets(input, INPUT_LEN + 1, stdin);
printf("%s\n", input);
```

```
int countNumSpaces(const char s[], int n)
{
    int counter = 0;

    for (int i = 0; i < n; i++)
    {
        if (s[i] == ' ')
        {
            counter++;
        }
    }

    return counter;
}

int countNumSpaces(const char s[])
{
    int counter = 0;

    for (int i = 0; s[i] != '\0'; i++)
    {
        if (s[i] == ' ')
        {
            counter++;
        }
    }

    return counter;
}
```

Slower

Wrong

How many space characters are in this string?



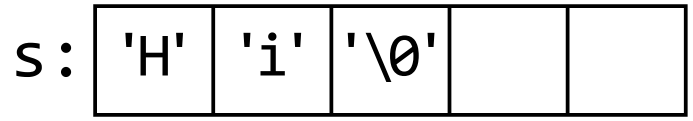
1?

2?

String Processing

- When you're *reading* characters from a string, you don't need the size of the array
- Instead, you care about the location of the `'\0'`

- When you're *writing* characters to a string, you do need the size of the array



`string.h`

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

```
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};
```

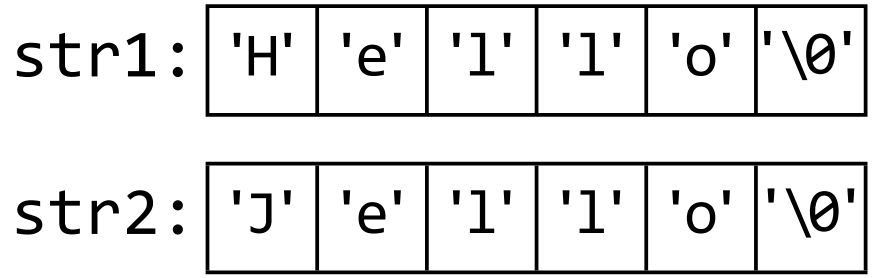
Error

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


str1 = str3;
```

Error

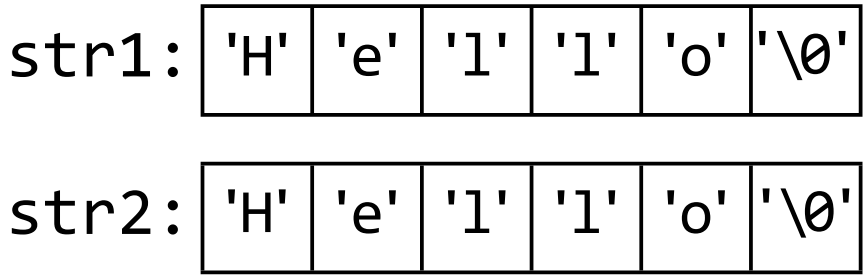
Strings
(not a recap)

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

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



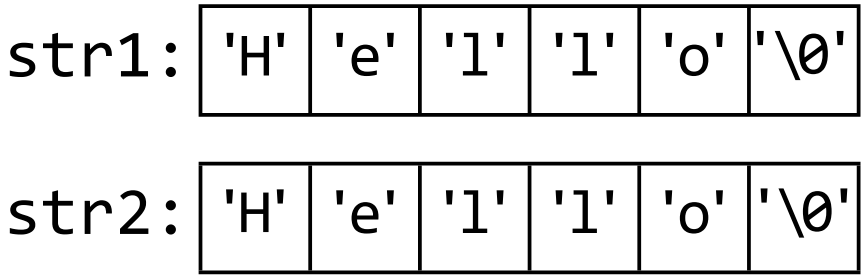
Different

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

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



Different

```
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

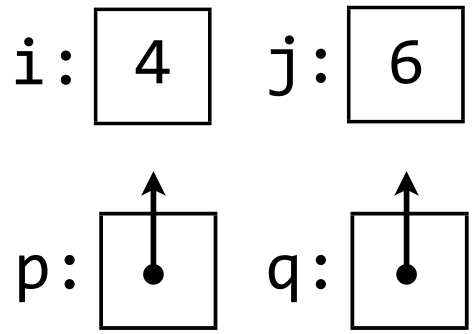
Different

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`



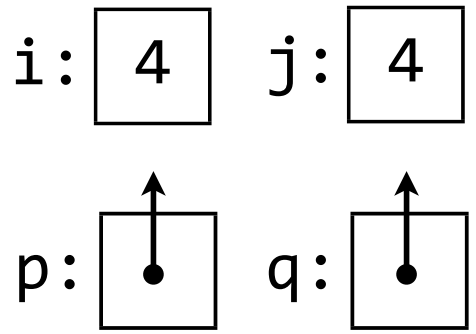
Equality

- 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

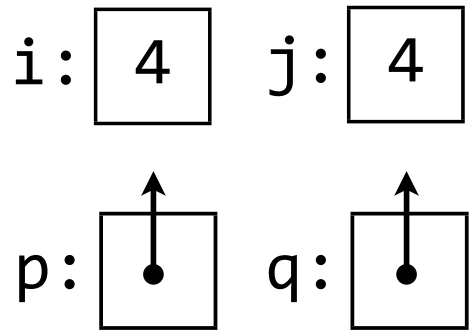
- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
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

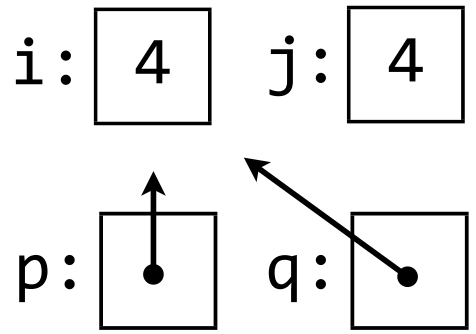
```
int i = 4;
int j = 6;

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);
```

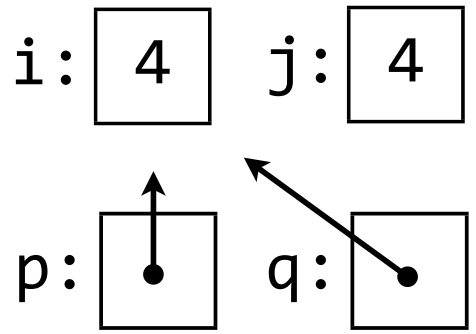
false

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

false

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

true



string.h

- strlen()
- strcpy()
- strcat()
- Many more

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);
length = strlen("");
```

5

0

Start at the first `char` in `s`

`while` (the current `char` is not `\0`)

{

  Add one to our length

  Move on to the next `char`

}

```
int my_strlen(const char *s)
{
    int n = 0;

    while (s[n] != '\0')
    {
        n++;
    }

    return n;
}
```

string.h

- Most functions take `char *` params
  - Can pass in an array
  - Can pass in a pointer to an array
  - Can pass in a literal
    - Be careful if it's not a `const char *` param!
- Make sure it's null terminated!

```
char s1[] = "Hello";
char *p = s1;
char s2[] = {'A', 'B'};
```

```
strlen(s1);
strlen(p);
strlen("Hello");
strlen(s2);
```

Wrong

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

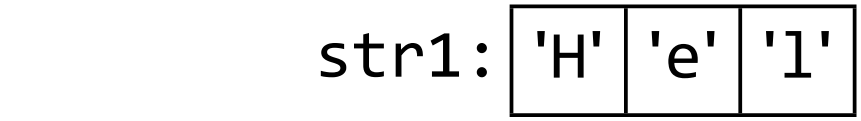
```
strcpy(str1, str2);
```

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

"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);
```



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`

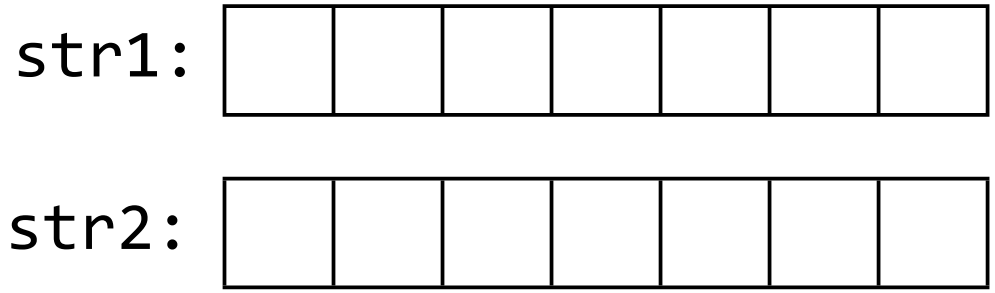
```
char str1[2 + 1];
char str2[MAX_LEN + 1] = "Hello";

strcpy(str1, str2);
```

Wrong

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

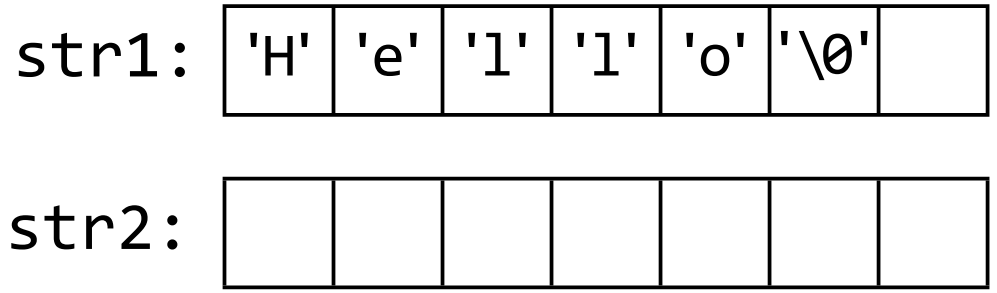
Why Only "Safer"?

```
#define MAX_LEN 6

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



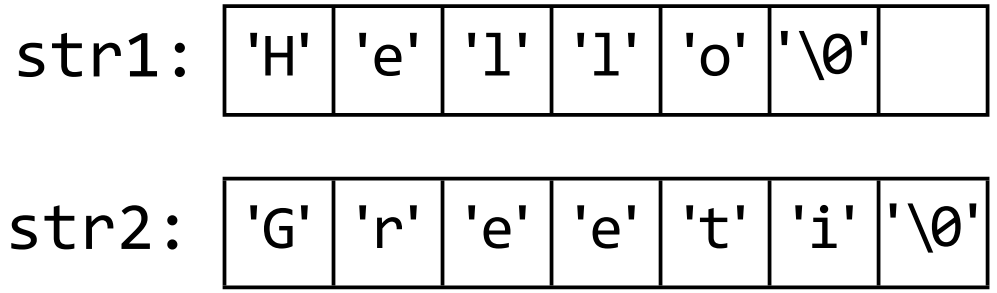
Why Only "Safer"?

```
#define MAX_LEN 6

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

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



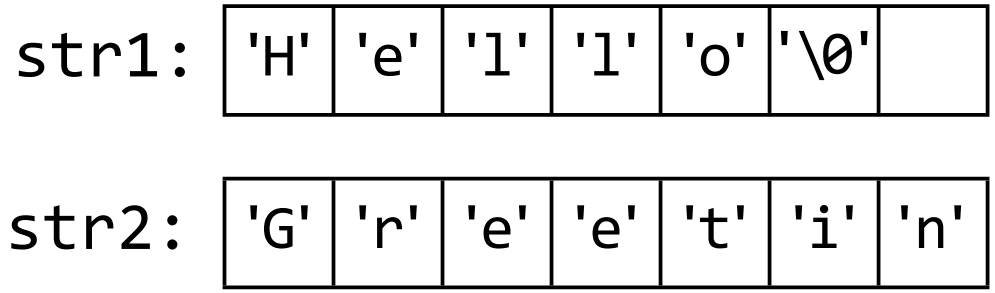
Why Only "Safer"?

```
#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"?

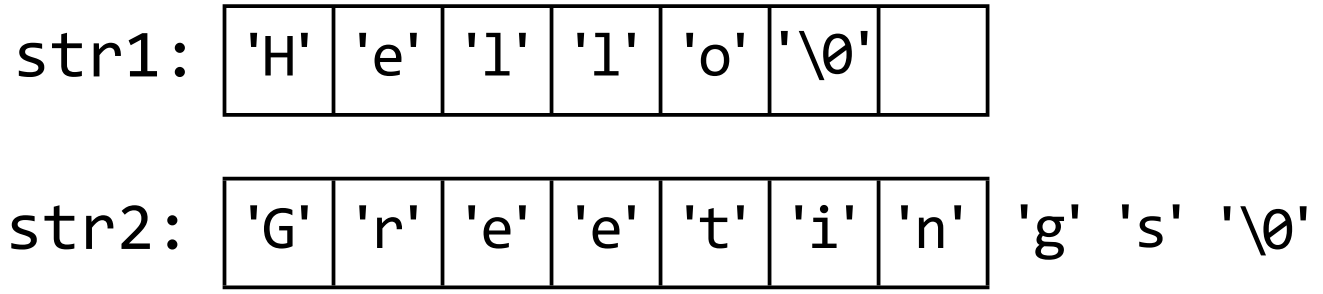
```
#define MAX_LEN 6

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

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

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



str2 is no longer a string

Why Only "Safer"?

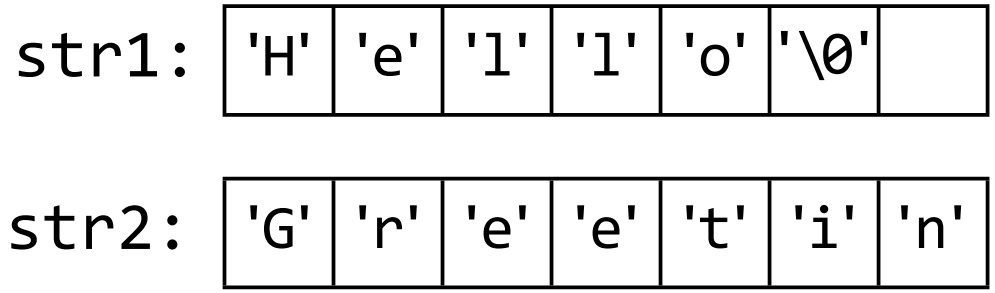
```
#define MAX_LEN 6

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"?

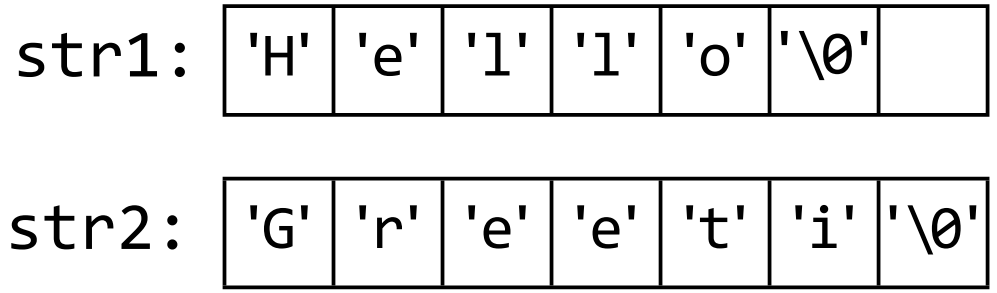
```
#define MAX_LEN 6

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

Start at the first char of s2

while (the current char is not \0)

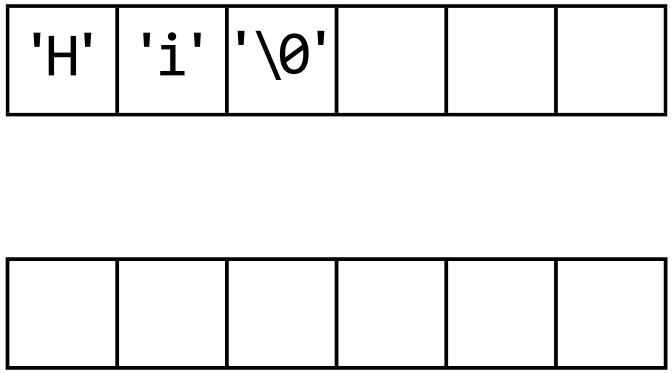
{

  Copy that character to s1

  Move on to the next char

}

Add a \0 at the current position in s1




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

Start at the first char of s2

while (the current char is not \0)

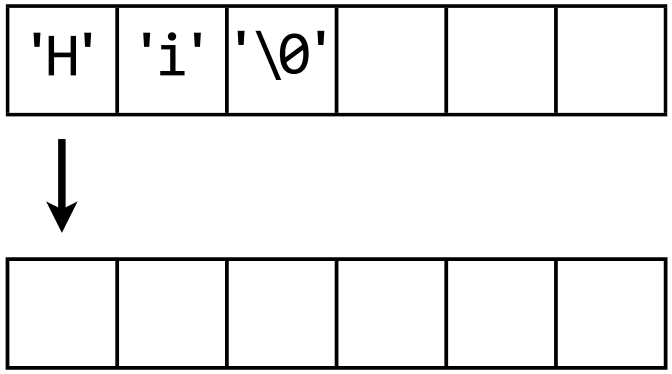
{

  Copy that character to s1

  Move on to the next char

}

Add a \0 at the current position in s1




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

Start at the first char of s2

while (the current char is not \0)

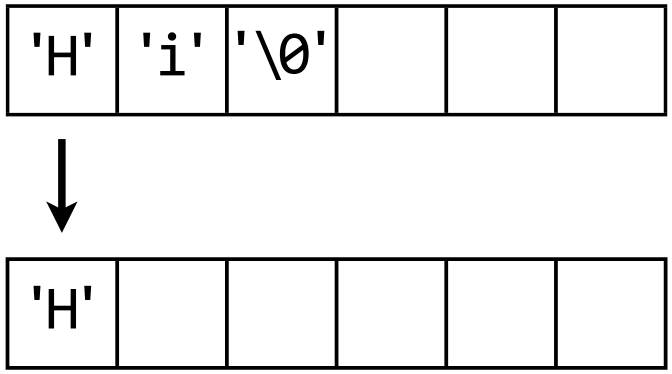
{

  Copy that character to s1

  Move on to the next char

}

Add a \0 at the current position in s1




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

Start at the first char of s2

while (the current char is not \0)

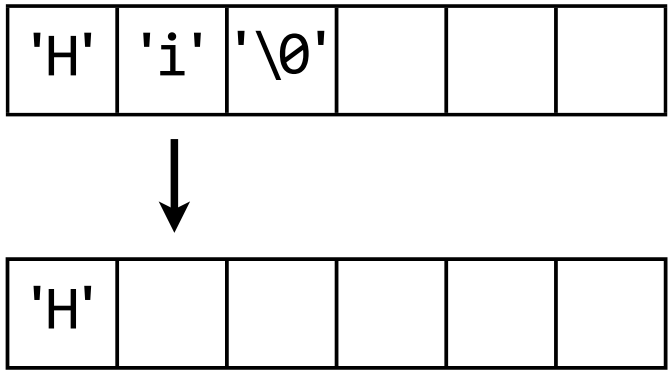
{

  Copy that character to s1

  Move on to the next char

}

Add a \0 at the current position in s1




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

Start at the first char of s2

while (the current char is not \0)

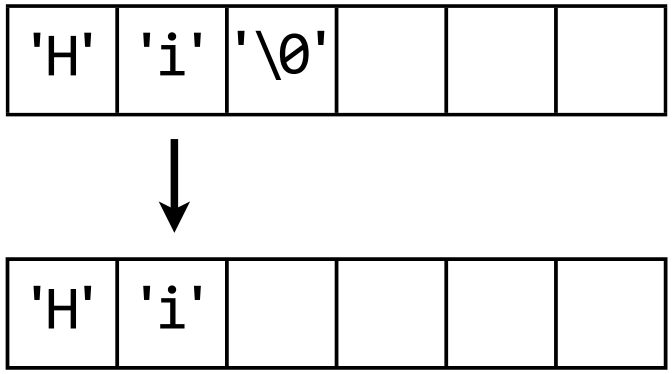
{

  Copy that character to s1

  Move on to the next char

}

Add a \0 at the current position in s1




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

Start at the first char of s2

while (the current char is not \0)

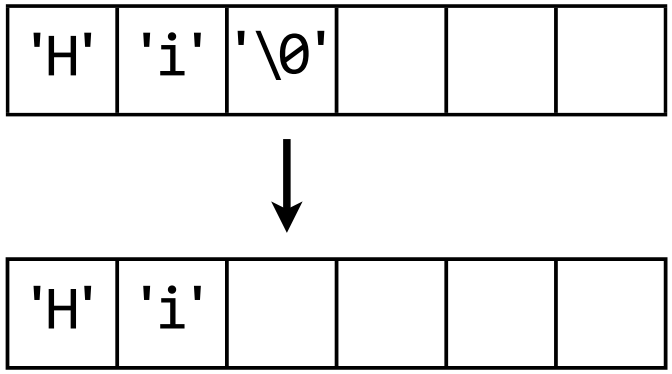
{

  Copy that character to s1

  Move on to the next char

}

Add a \0 at the current position in s1




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

Start at the first char of s2

while (the current char is not \0)

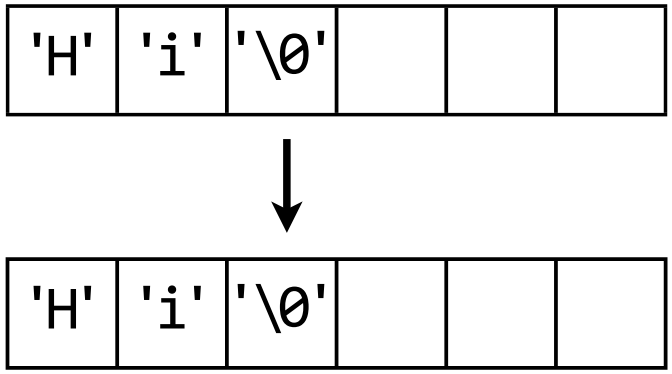
{

  Copy that character to s1

  Move on to the next char

}

Add a \0 at the current position in s1




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

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

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

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

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