CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 8.1: Strings

Toronto Metropolitan University

Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

Today

Strings

Character Arrays

`#include <string.h>`

Operations on Strings

S1 T1 R1 I1 N1 G2 S1

Recall: Arrays

An array is a sequence of values of the same type:

```
#include <stdio.h>

int main()
{
    int nums[100];
    return 0;
}
```

An array of 100 integers

Character Arrays

That type can, of course, be **char**:

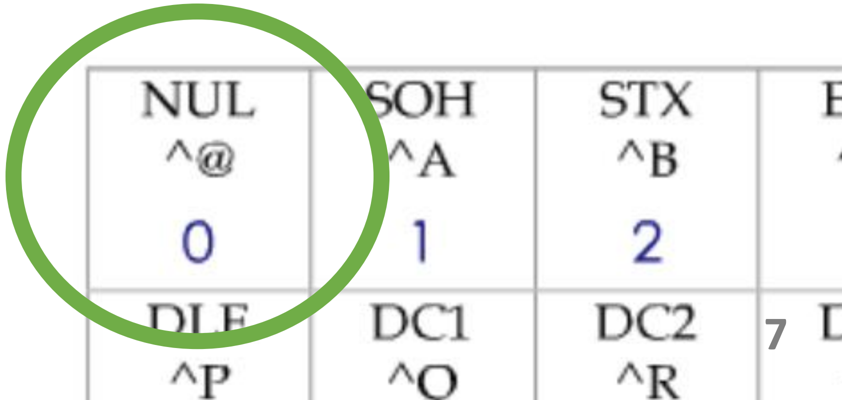
```
#include <stdio.h>

int main()
{
    char word[15];
    return 0;
}
```

An array of 15 characters

Strings

A **string** is a **character array** that is terminated by the ***null character*** – ‘\0’

```
#include <stdio.h>
int main()
{
    char city[] = {‘T’,‘o’,‘r’,’o’,’n’,’t’,’o’,’\0’};
    printf(“%d\n”, ‘\0’); /* 0 on the asci table */
    return 0;
}
```



NUL ^@ 0	SOH ^A 1	STX ^B 2	E
DLE ^P	DC1 ^Q	DC2 ^R	D

String Initialization

```
#include <stdio.h>
int main()
{
      char city[] = “Toronto”;
      return 0;
}
```

What is the size of city?

When we initialize this way, the null character is automatically added to the end of the char array.

String size equals the number of characters plus one.

'\0'

Double Quotes

```
#include <stdio.h>
int main()
{
      char city[] = “Toronto”;
      return 0;
}
```

Double quotes specifies a string. What do single quotes mean?

```
char letter = ‘A’;
```

Double Quotes

Single quotes denote individual characters:

`‘c’, ‘?’, ‘\n’, ‘7’, ‘\0’`

Double quotes indicate a string (char array ending in **`‘\0’`**)

`“CPS188”, “Ryerson”, “A”`

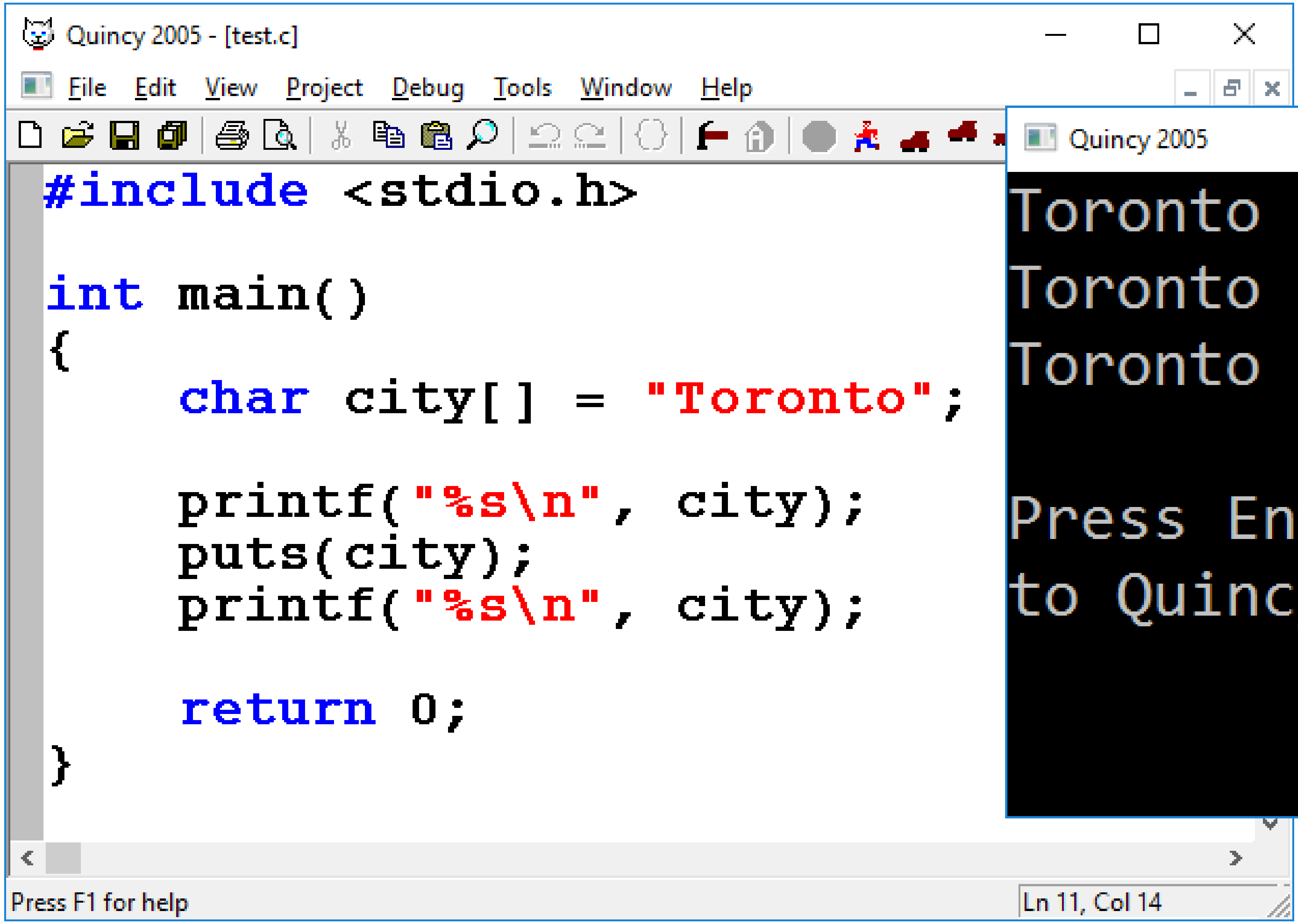
A string can be initialized with one character!

Printing Strings

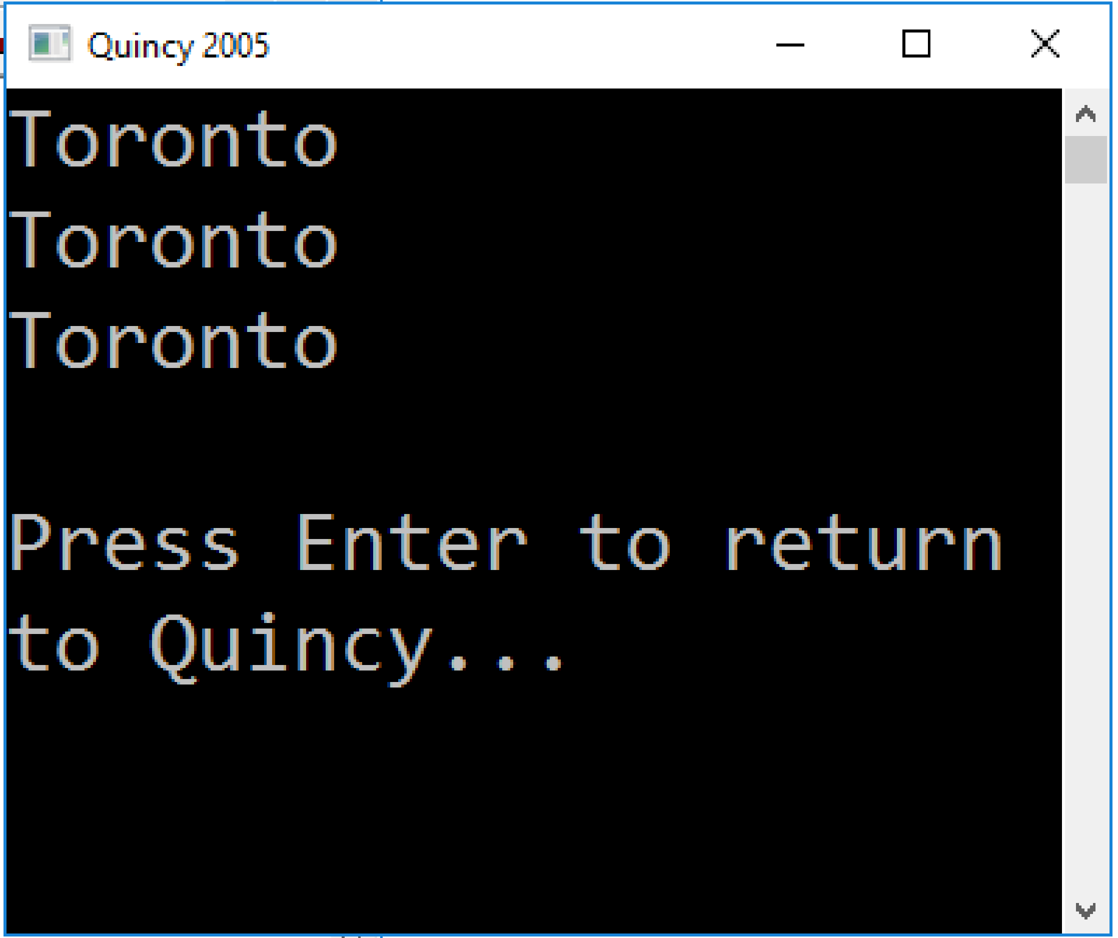
```
#include <stdio.h>

int main()
{
      char city[] = “Toronto”;

      printf(“%s”, city);

      puts(city);

      return 0;
}
```

Placeholder for string

puts is for strings only. Inserts a newline after printing the string.



```c
#include <stdio.h>

int main()
{
    char city[] = "Toronto";

    printf("%s\n", city);
    puts(city);
    printf("%s\n", city);

    return 0;
}
```



```
Toronto
Toronto
Toronto

Press Enter to return
to Quincy...
```

How Does `printf` Know…

```
printf(“%s”, city);
```

… when it’s at the end of the string?

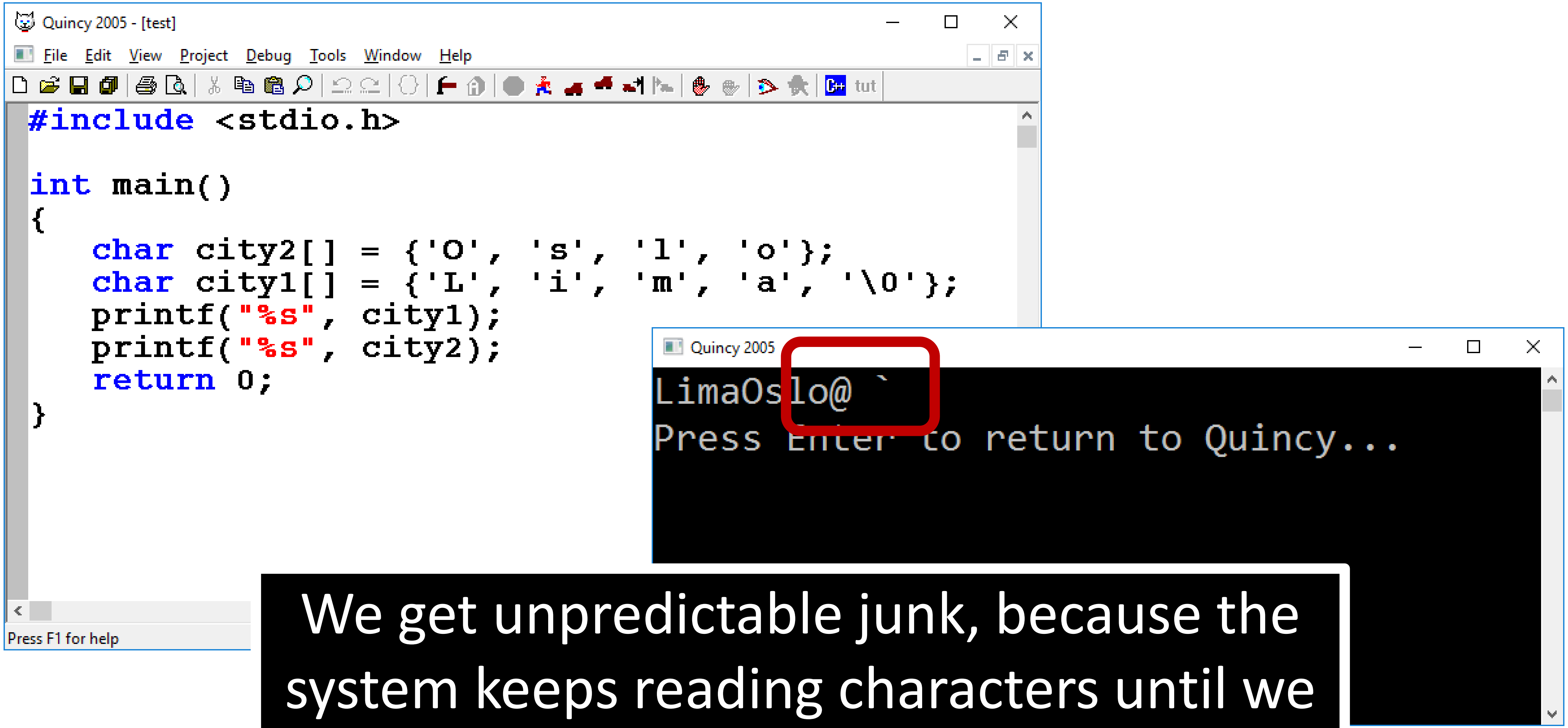
`city` is just an address, like any other array.
It doesn’t tell the system how long the string is.

When we print a string using the **%s** placeholder, it tells the system to keep printing characters until we hit the null character, **‘\0’**

```
#include <stdio.h>

int main()
{
    char city1[] = {‘L’, ‘i’, ‘m’, ‘a’, ‘\0’};
    char city2[] = {‘O’, ‘s’, ‘l’, ‘o’};
    printf(“%s”, city1);
    printf(“%s”, city2);
    return 0;
}
```

No null character!

What happens?



```c
#include <stdio.h>

int main()
{
    char city2[] = {'O', 's', 'l', 'o'};
    char city1[] = {'L', 'i', 'm', 'a', '\0'};
    printf("%s", city1);
    printf("%s", city2);
    return 0;
}
```

```
LimaOslo@ `
Press Enter to return to Quincy...
```

We get unpredictable junk, because the system keeps reading characters until we hit '\0' somewhere else.

Accessing String Elements

```
#include <stdio.h>

int main()
{
      char city[] = “Toronto”;

      city[1] = ‘a’;
      city[3] = ‘a’;
      city[6] = ‘a’;

      printf(“%s”, city);

      return 0;
}
```

Console:

```
Taranta
```

```
int i;
char city[] = “Toronto”;

for (i = 0; i < 7; i++)
{
    if (city[i] == ‘o’)
    {
        city[i] = ‘a’;
    }
}

printf(“%s”, city);
```

Loop through each character in string

Replace ‘o’ with ‘a’

Console:

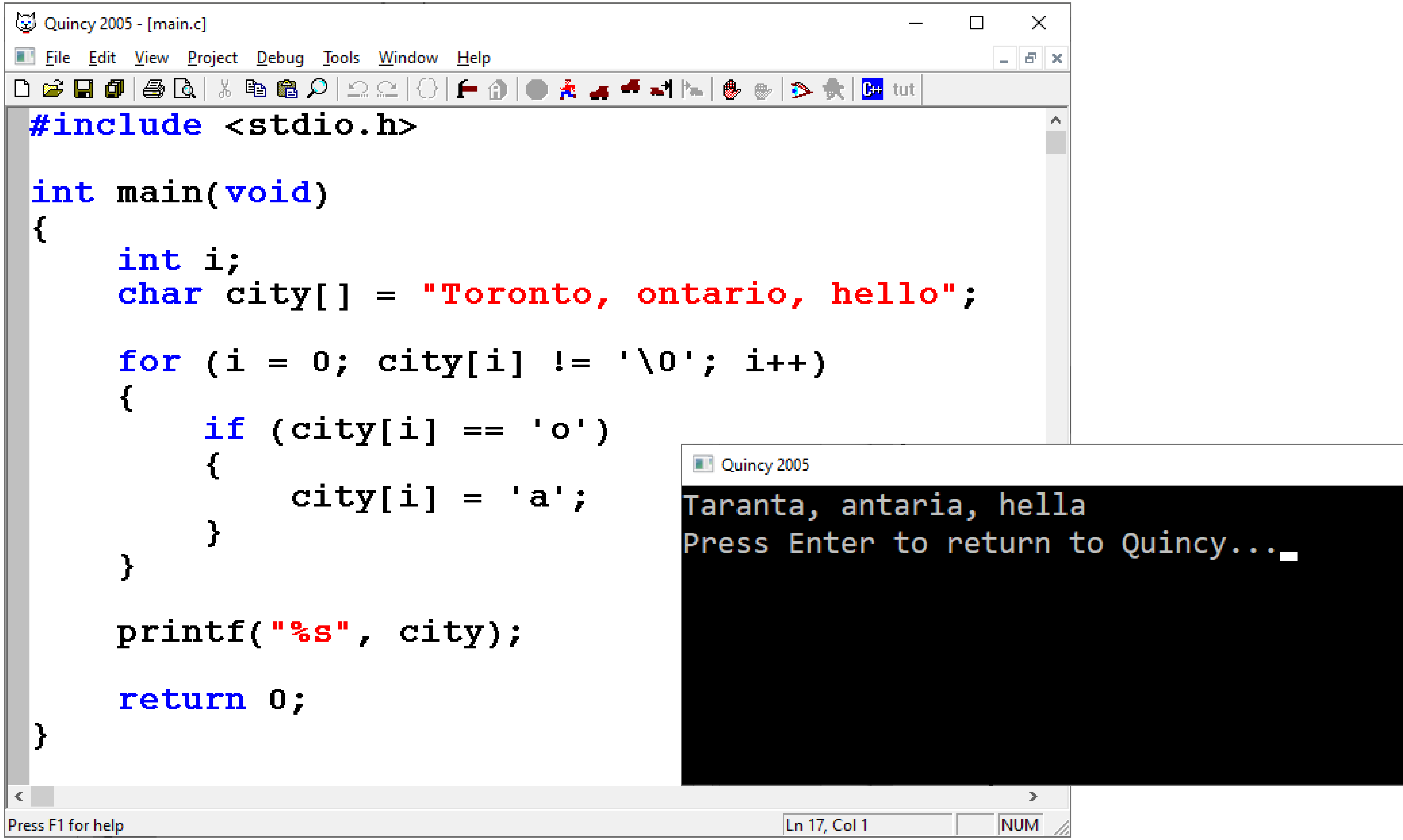
```
Taranta
```

```
int i;
char city[] = “Toronto”;

for (i = 0; city[i] != ‘\0’; i++)
{
      if (city[i] == ‘o’)
      {
            city[i] = ‘a’;
      }
}

printf(“%s”, city);
```

Even better!
- Go until we hit the null character.
- Works on any string

Console:

```
Taranta
```



```c
#include <stdio.h>

int main(void)
{
	int i;
	char city[] = "Toronto, ontario, hello";

	for (i = 0; city[i] != '\0'; i++)
	{
		if (city[i] == 'o')
		{
			city[i] = 'a';
		}
	}

	printf("%s", city);

	return 0;
}
```

```
Taranta, antaria, hella
Press Enter to return to Quincy...
```

Strings as Input

```
char city[10];

scanf("%s", city);
```

Notice something missing?

Array name alone is an ADDRESS

```
scanf("%s", &city[0]); /* Equivalent */
```

```c
char city[16];

scanf("%s", city);
printf("%s", city);
```

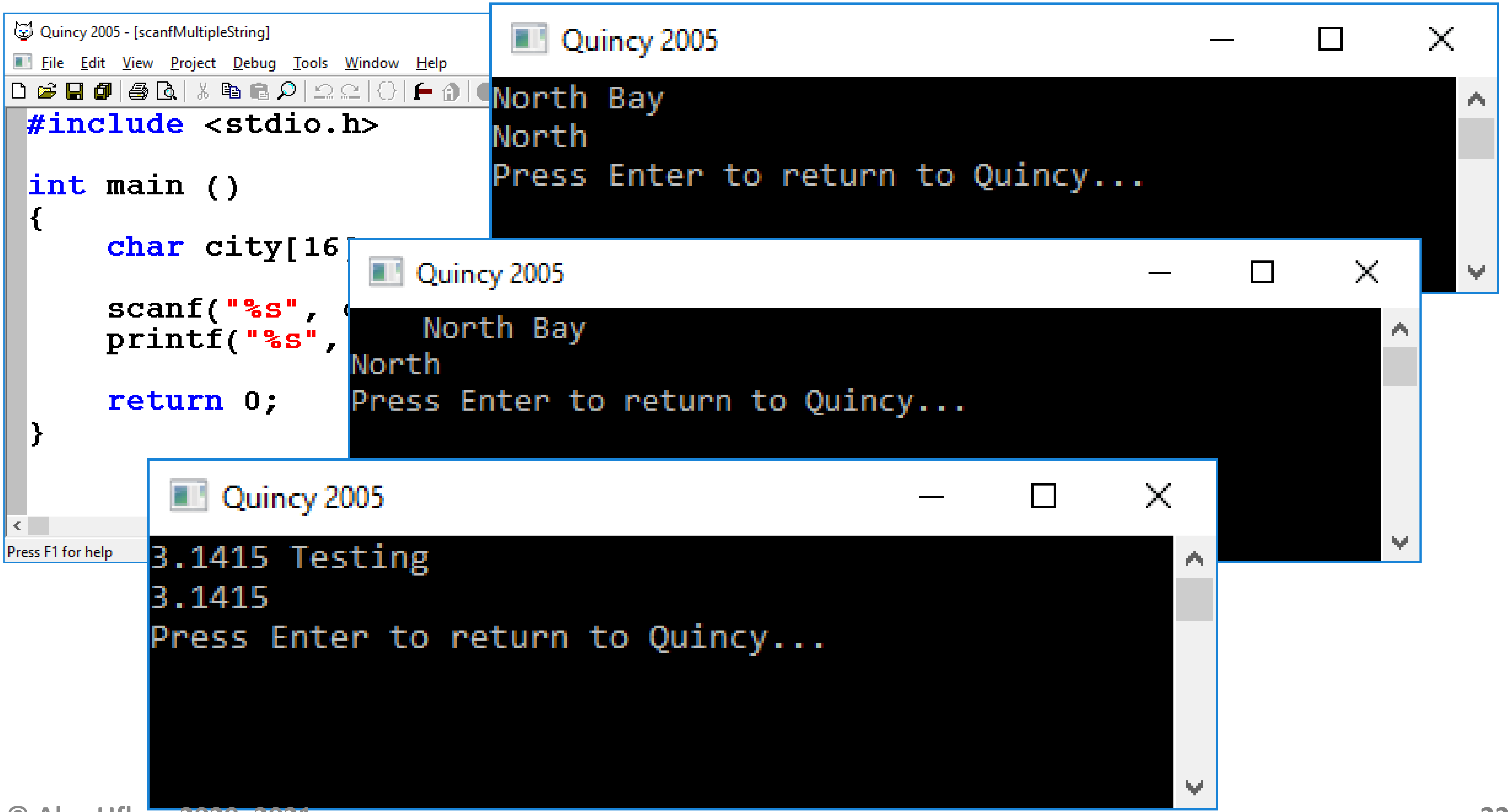
We only scan one string

Space in input is considered a delimiter

Console

```
North Bay
North
```

Treated as two separate strings


Quincy 2005 - [scanfMultipleString]
File Edit View Project Debug Tools Window Help
Press F1 for help
Quincy 2005

```
#include <stdio.h>

int main ()
{
      char city[16

      scanf("%s", 
      printf("%s",

      return 0;
}
```

```
North Bay
North
Press Enter to return to Quincy...
```

```
    North Bay
North
Press Enter to return to Quincy...
```

```
3.1415 Testing
3.1415
Press Enter to return to Quincy...
```

```
char city[16];
gets(city); // Use fgets for files
printf("%s", city);
```

Newline character is considered a delimiter

Console

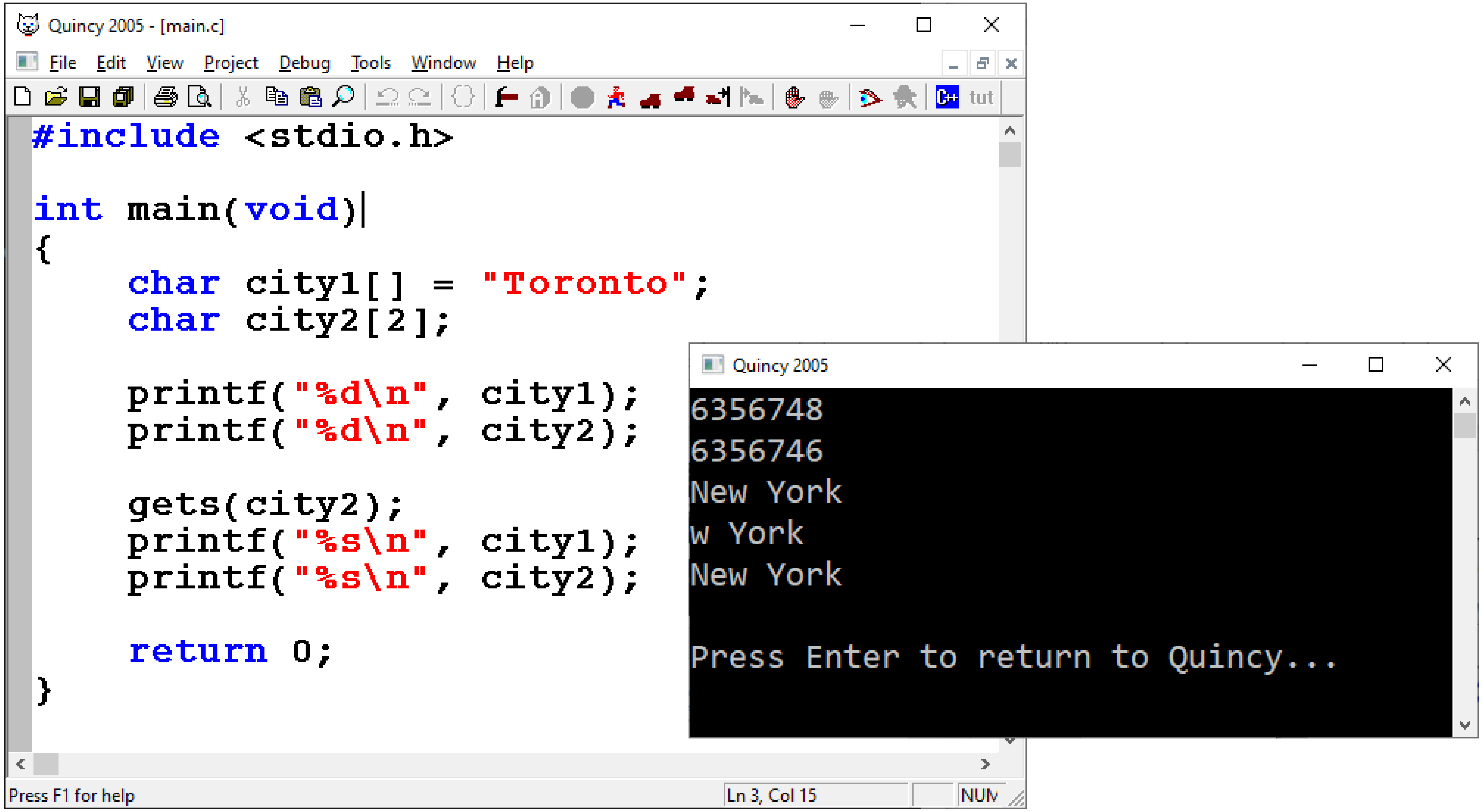
```
North Bay
North Bay
```

Be Careful…

```
char city[16];
gets(city);
printf("%s", city);
```

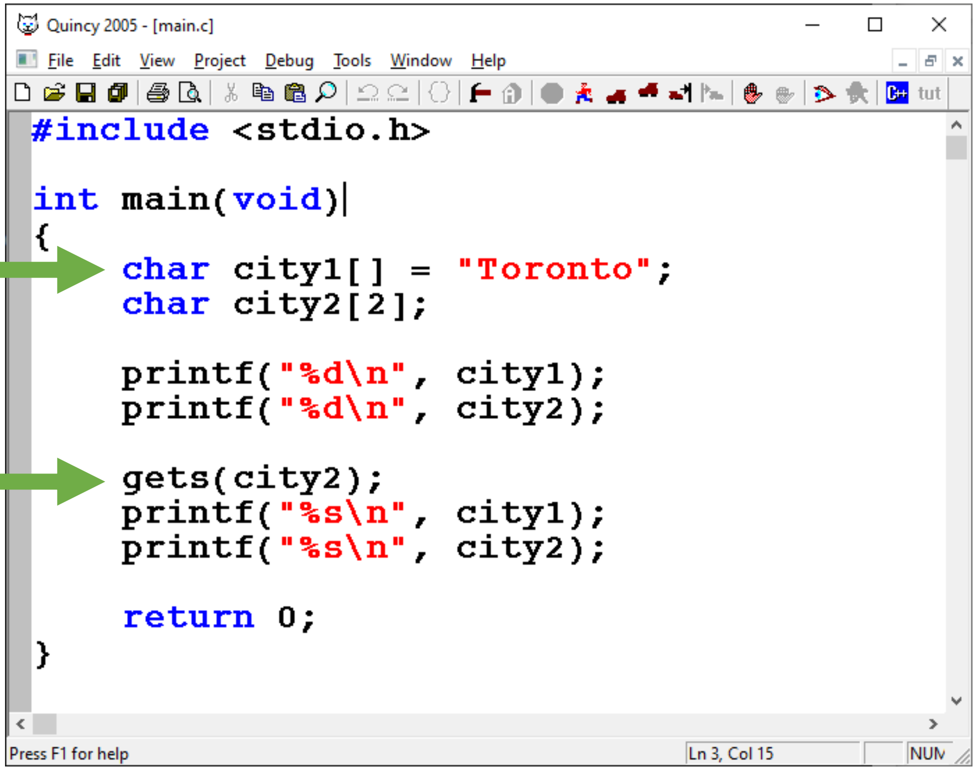
- What if the user enters string longer than 15 characters?
- 15 + `'\0'` = 16 total

- We will overrun the bounds of our array!
- Just like integer arrays, double arrays, etc.
- Be sure to declare a char array long enough to fit the input.

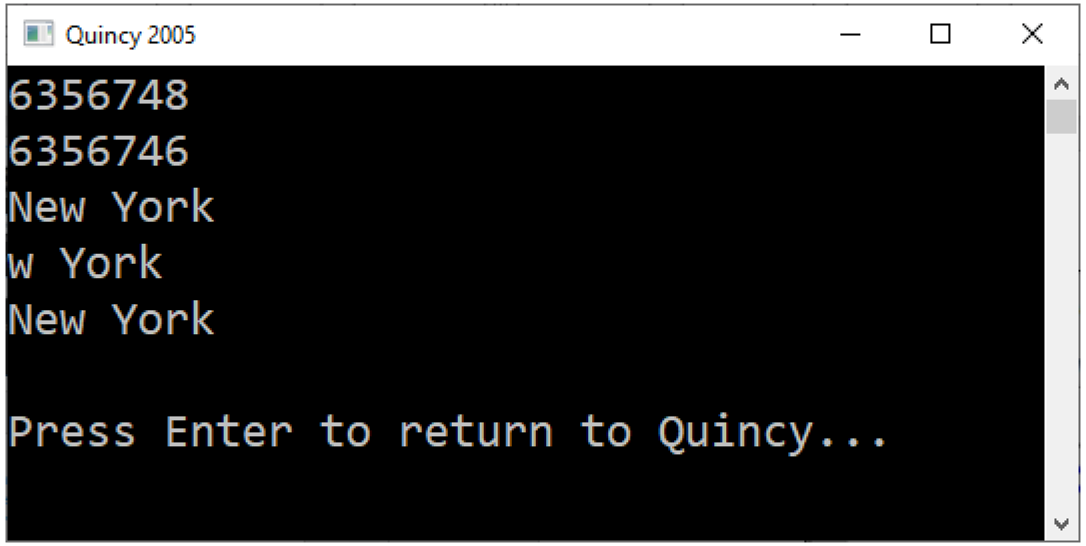


```
#include <stdio.h>

int main(void)
{
    char city1[] = "Toronto";
    char city2[2];

    printf("%d\n", city1);
    printf("%d\n", city2);

    gets(city2);
    printf("%s\n", city1);
    printf("%s\n", city2);

    return 0;
}
```

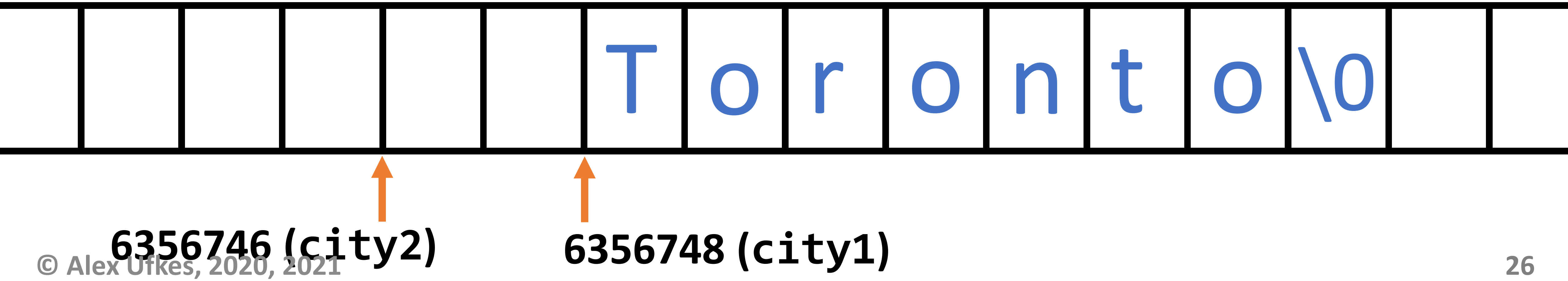
```
6356748
6356746
New York
w York
New York

Press Enter to return to Quincy...
```

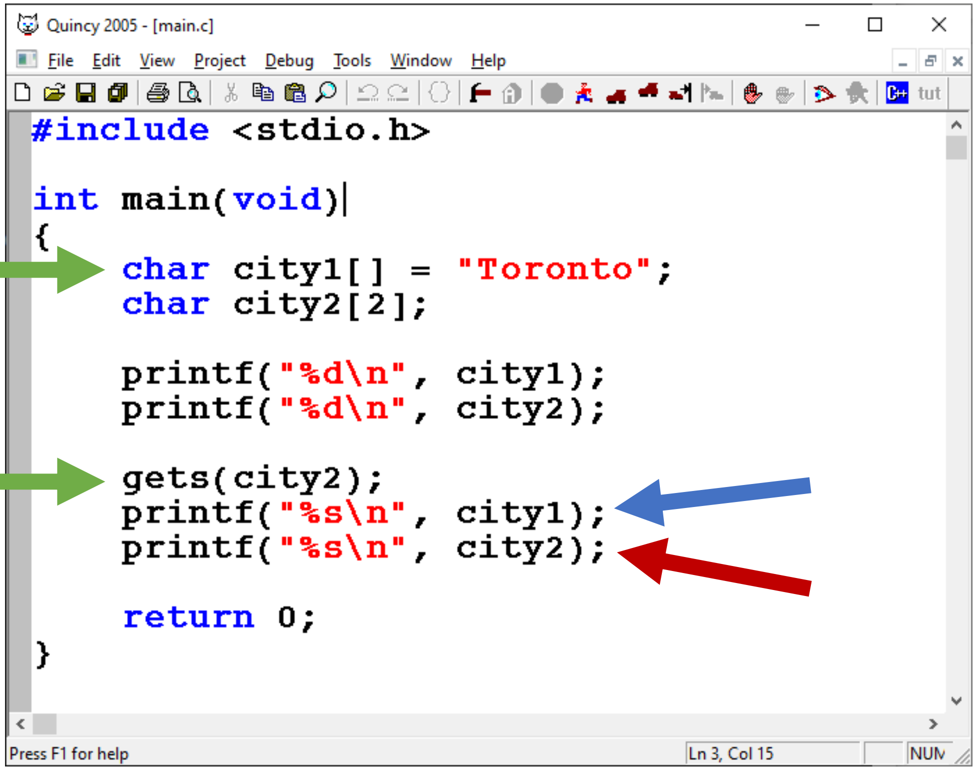


```c
#include <stdio.h>

int main(void)
{
    char city1[] = "Toronto";
    char city2[2];

    printf("%d\n", city1);
    printf("%d\n", city2);

    gets(city2);
    printf("%s\n", city1);
    printf("%s\n", city2);

    return 0;
}
```

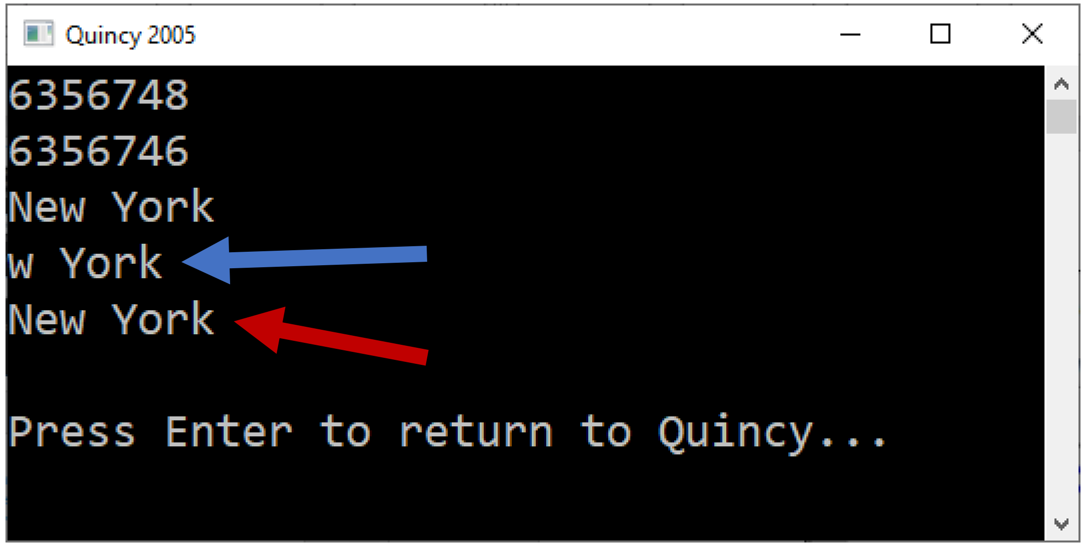


```
6356748
6356746
New York
w York
New York

Press Enter to return to Quincy...
```

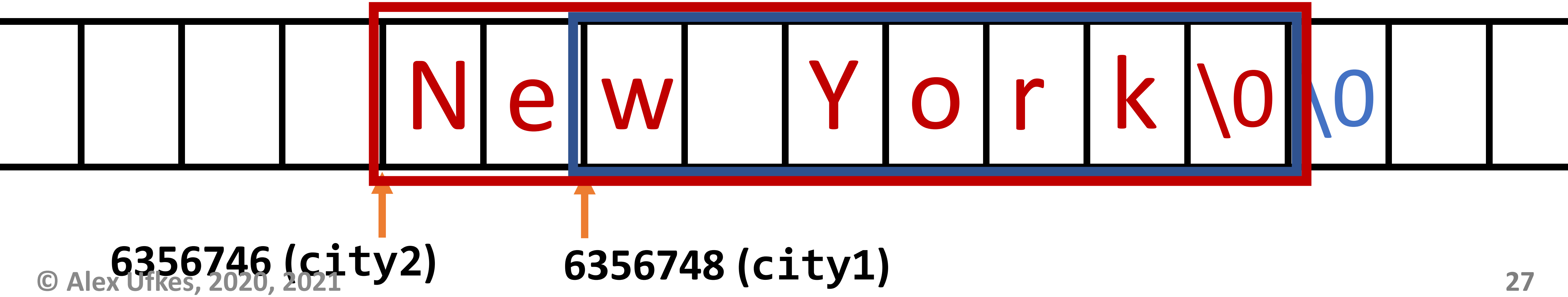




```c
#include <stdio.h>

int main(void)
{
    char city1[] = "Toronto";
    char city2[2];

    printf("%d\n", city1);
    printf("%d\n", city2);

    gets(city2);
    printf("%s\n", city1);
    printf("%s\n", city2);

    return 0;
}
```



```
6356748
6356746
New York
w York
New York

Press Enter to return to Quincy...
```



String Functions

```c
#include <string.h>

char city[] = “Oslo”;
int length = strlen(city);
printf(“Length: %d\n”, length);
```

Console

```
Length: 4
```

Length does not include the null character!

String Copy

```
char city1[8] = “Toronto”;
char city2[8];
city2 = city1;
```

ILLEGAL!

Use `strcpy` instead

strcpy

Found in **string.h**

```
char city1[8] = “Toronto”;
char city2[8];

strcpy(city2, city1)
```

Copies string **city1** into string **city2**.

String Copy

```
char city[8];
city = “Markham”;
```

Also ILLEGAL!

A string can be initialized **ONLY** upon declaration. Otherwise….

Use `strcpy` instead

strcpy

Found in `string.h`

```
char city[8];

strcpy(city, “Markham”);
```

Copies string literal `“Markham”` into **`city`**.

`strcpy` copies *right* string into *left* string

strcpy

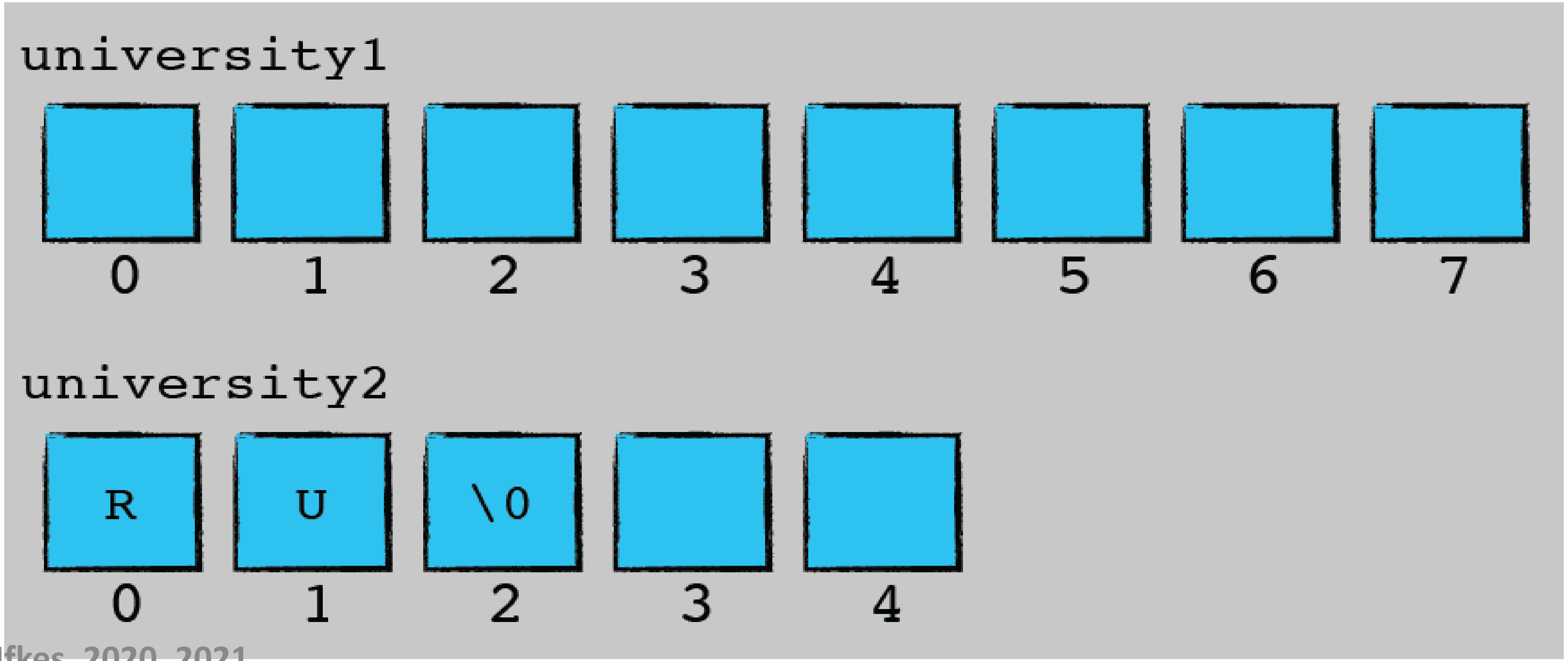
```
char university1[8];
char university2[8];

strcpy(university1, "Ryerson");
strcpy(university2, university1);
```

Make sure character arrays are big enough!

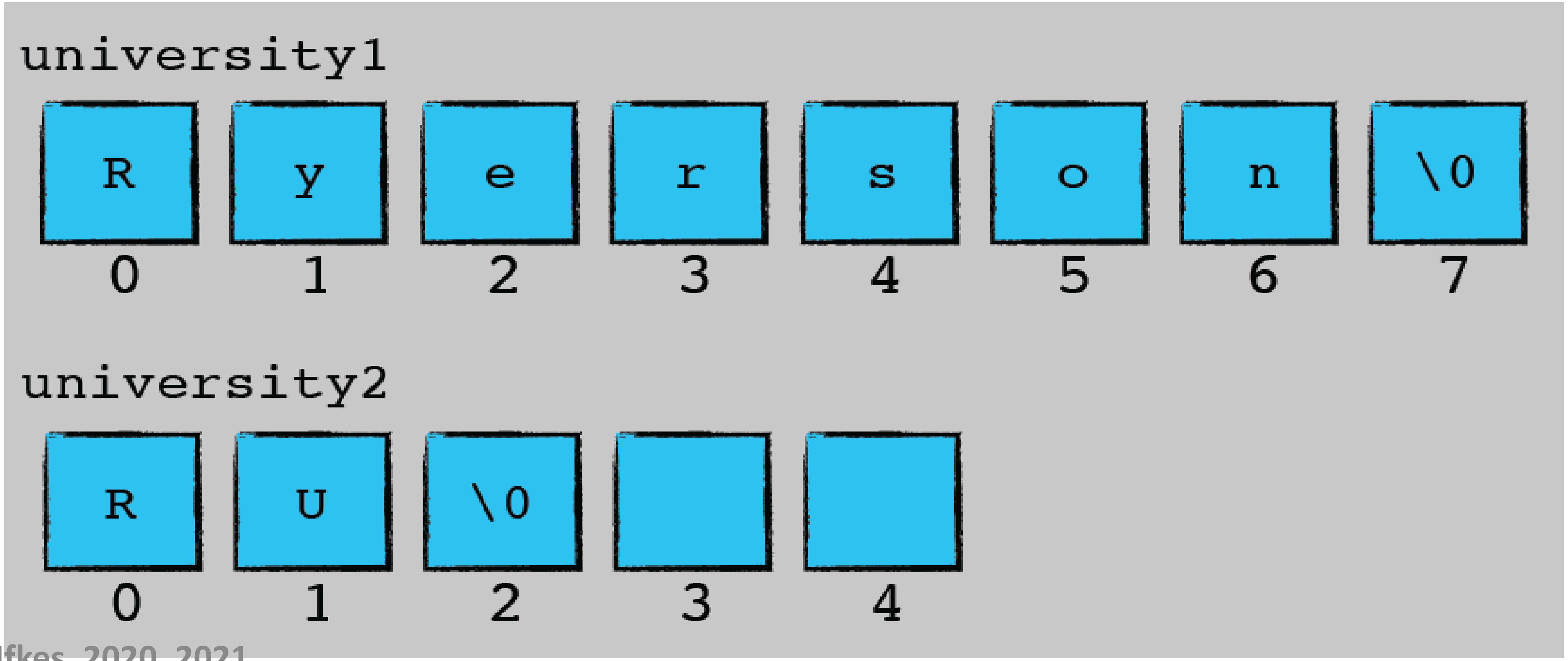
Copies string literal "Ryerson" into university1.

Copies string university1 into university2.

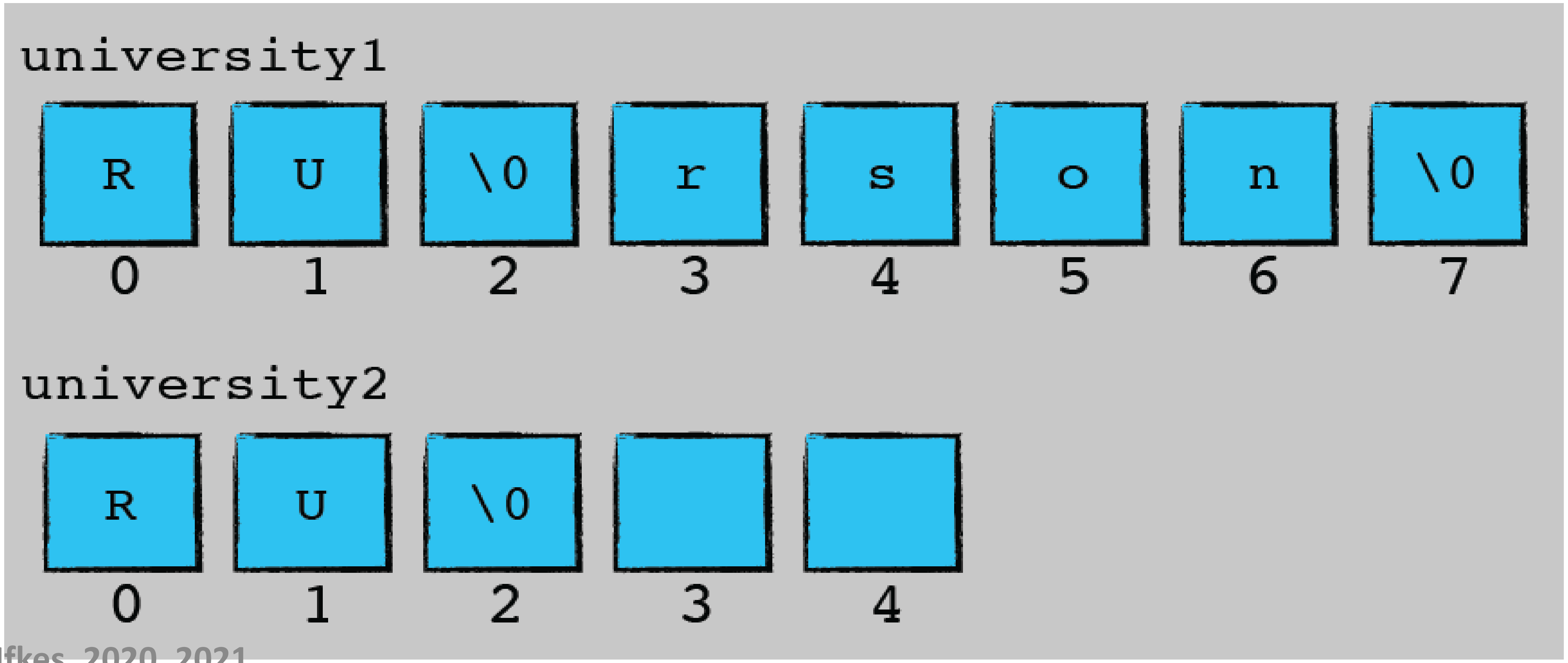
```
char university1[8];
char university2[5] = "RU";
strcpy(university1, "Ryerson");
strcpy(university1, university2);
```

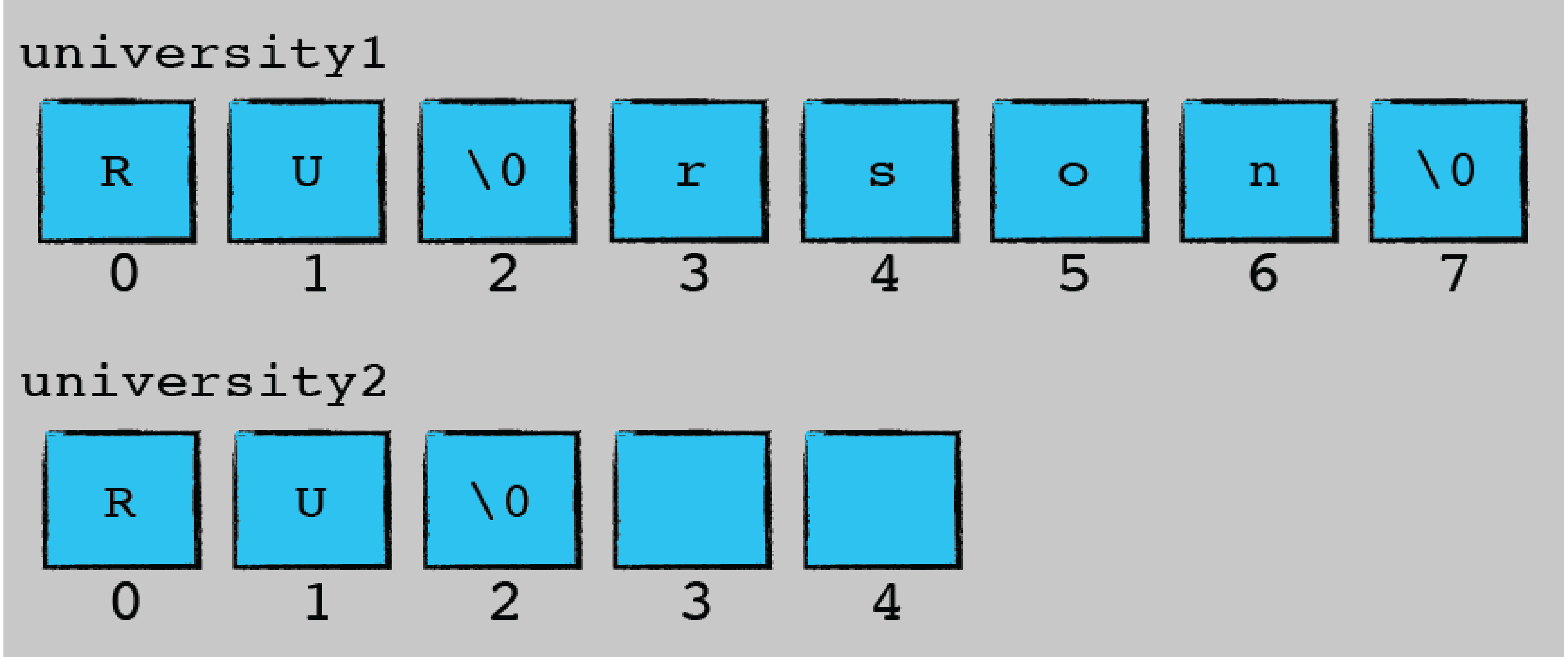


```
char university1[8];
char university2[5] = "RU";
strcpy(university1, "Ryerson");
strcpy(university1, university2);
```



```
char university1[8];
char university2[5] = "RU";
strcpy(university1, "Ryerson");
strcpy(university1, university2);
```





```
puts(university1);
puts(university2);
```

Console:

```
RU
RU
```

Strings to Numbers

`atoi()` & `atof()`

Defined in **`stdlib.h`**

`atoi()`

Convert string to **integer**: `int x = atoi(“17”);`

`atof()`

Convert string to **double**: `double x = atof(“3.14”);`

Strings to Numbers

`atoi()` & `atof()`

Each function parses the string until a character that **doesn't make sense** is found.

`atoi()` `int x = atoi("17.89");`
Hits the decimal and stops, giving 17

`atof()` `double x = atof("17.89qqq");`
Hits the `'q'` character and stops, giving 17.89

+, - symbols are fine, as is scientific notation for floating point: 17.89e8

Arrays of

Array of Strings

```
#include <stdio.h>

int main()
{
    char months[12][10];

    return 0;
}
```

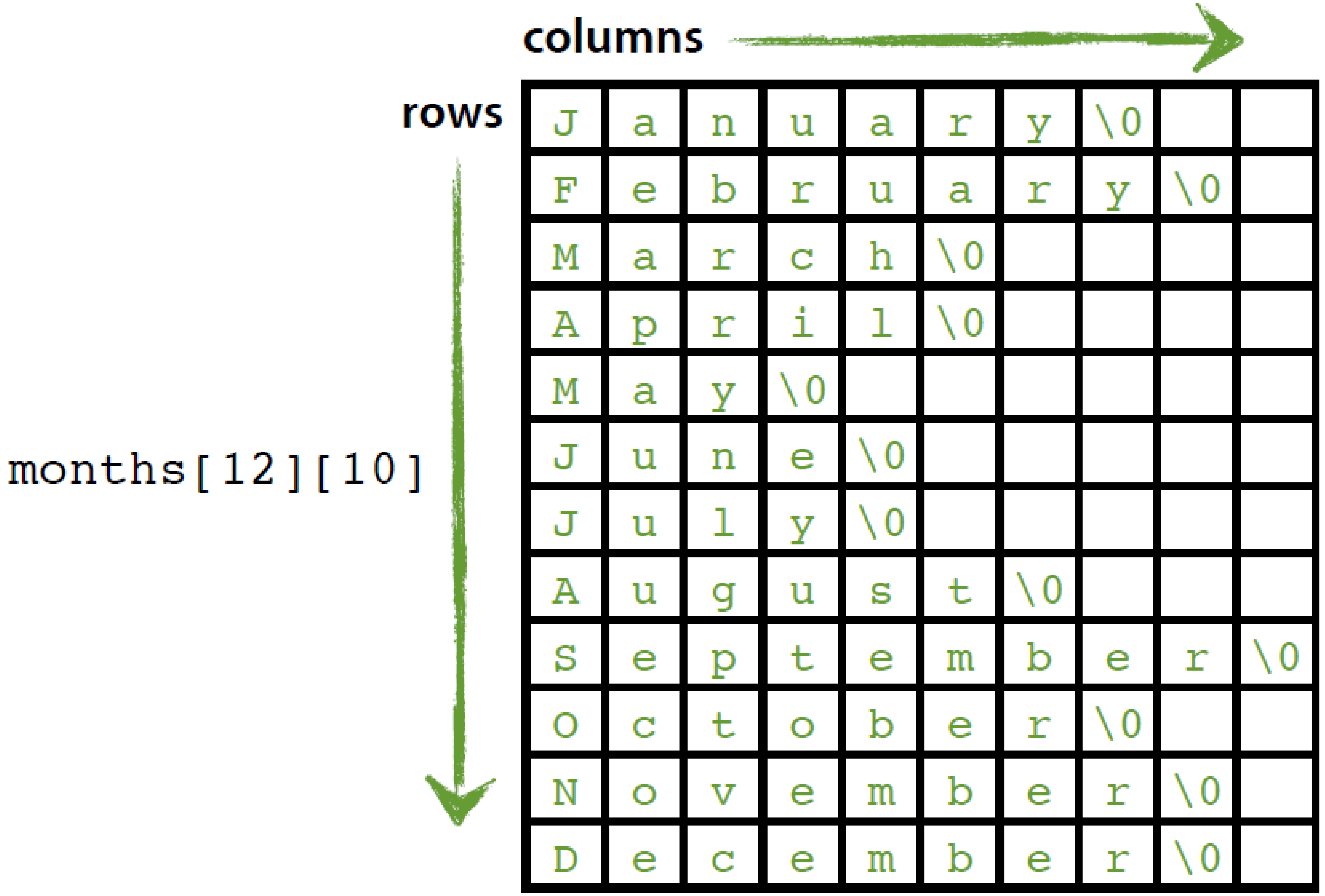
Number of strings

Maximum length of each string

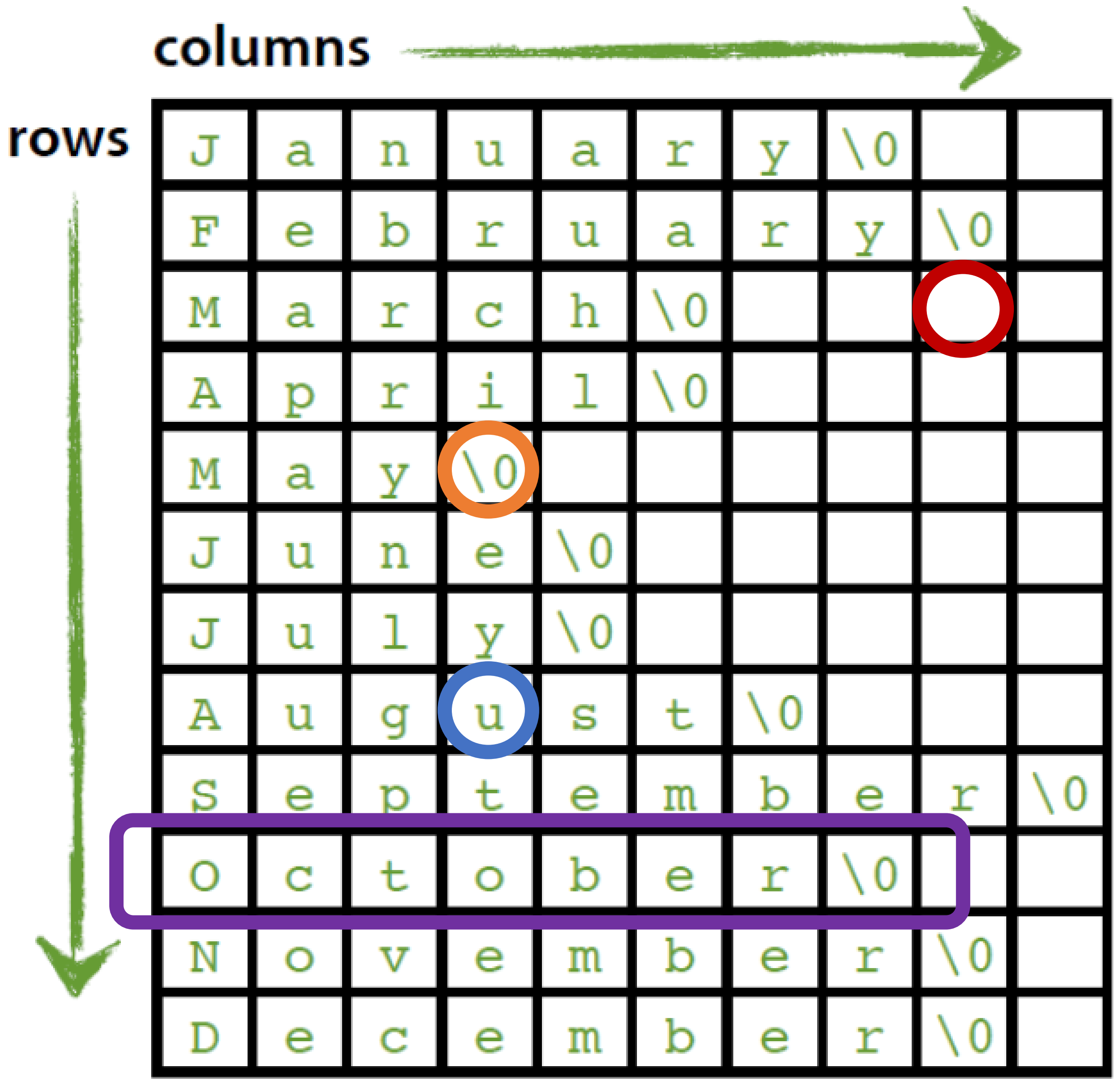
Array of Strings

Initialize using literals:

```
char months[12][10] =
    { “January”, “February”, “March”,
      “April”,   “May”,      “June”,
      “July”,    “August”,   “September”,
      “October”, “November”, “December” };
```



J	a	n	u	a	r	y	\0		
F	e	b	r	u	a	r	y	\0	
M	a	r	c	h	\0				
A	p	r	i	l	\0				
M	a	y	\0						
J	u	n	e	\0					
J	u	l	y	\0					
A	u	g	u	s	t	\0			
S	e	p	t	e	m	b	e	r	\0
O	c	t	o	b	e	r	\0		
N	o	v	e	m	b	e	r	\0	
D	e	c	e	m	b	e	r	\0	



J	a	n	u	a	r	y	\0		
F	e	b	r	u	a	r	y	\0	
M	a	r	c	h	\0				
A	p	r	i	l	\0				
M	a	y	\0						
J	u	n	e	\0					
J	u	l	y	\0					
A	u	g	u	s	t	\0			
S	e	p	t	e	m	b	e	r	\0
O	c	t	o	b	e	r	\0		
N	o	v	e	m	b	e	r	\0	
D	e	c	e	m	b	e	r	\0	

Value of months[7][3]?

Value of months[4][3]?

Value of months[2][8]?

The memory is ours, but we haven’t assigned it a value

Value of months[9]?

Address of the first character of October - **&months[9][0]**

```
printf(“%s”, months[9]);
```

Console

```
January
February
March
April
May
June
July
August
September
October
November
December
```

```
char months[12][10] =
   {“January”, “February”, “March”,
    “April”,   “May”,      “June”,
    “July”,    “August”,   “September”,
    “October”, “November”, “December”};
```

```
for (i = 0; i < 12; i++)
   printf(“%s\n”, months[i]);
```

More String Functions: `strncpy`

```
char *strncpy(char *dest, const char *src, size_t n)
```

Like **`strcpy`**, but only copies n characters.

`dest`	Pointer to the destination string.
`src`	• Pointer to the source string. • **const?** can't modify it. Strncpy won't modify src.
`n`	• Number of characters to copy. • **size_t?** Alias for unsigned integer.

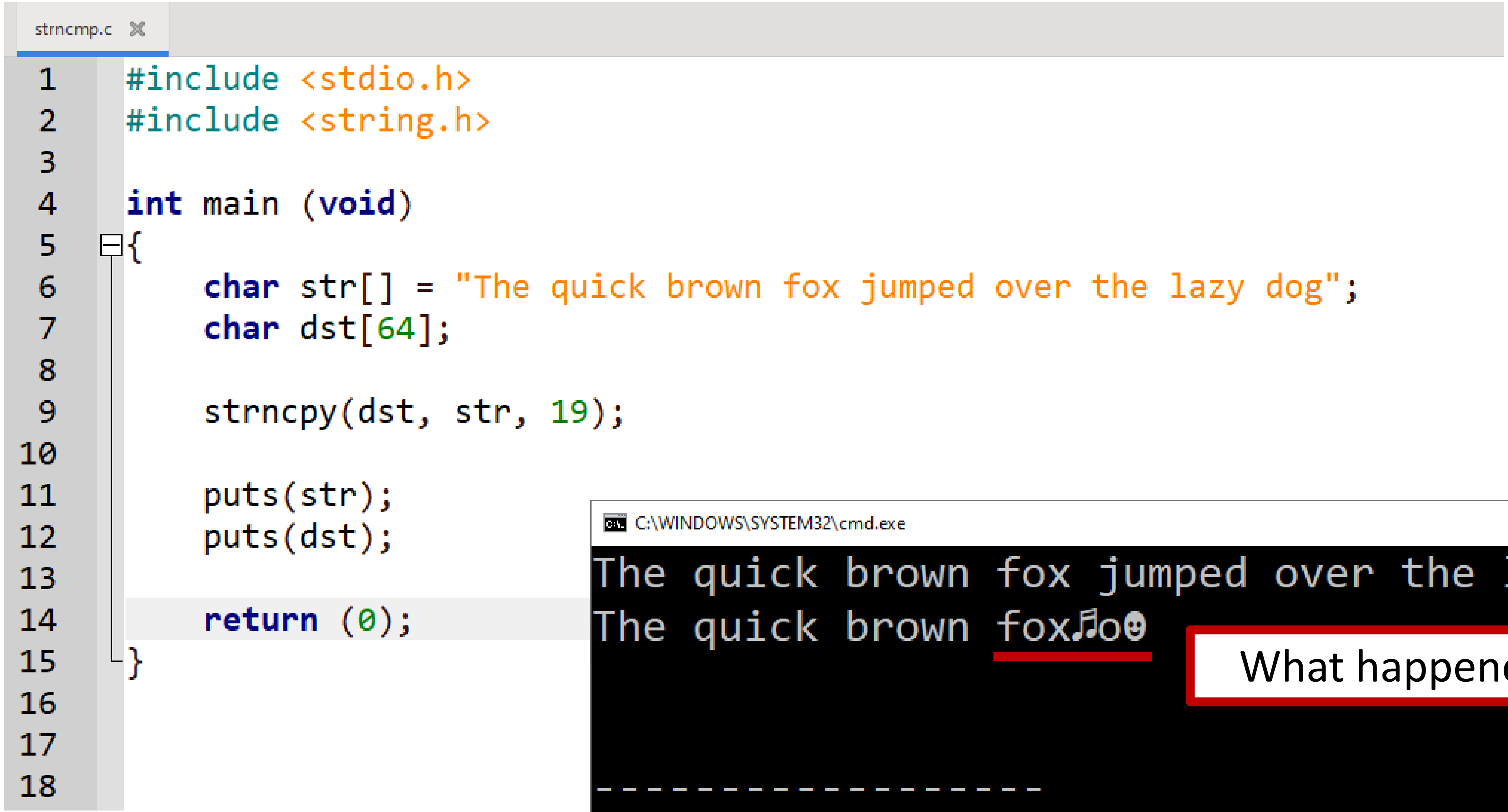
Returns a pointer to the copied string (`dest`)

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

int main (void)
{
  char str[] = "The quick brown fox jumped over the lazy dog";
  char dst[64];

  strncpy(dst, str, 19);

  puts(str);
  puts(dst);

  return (0);
}
```

Make sure **dst** is large enough!

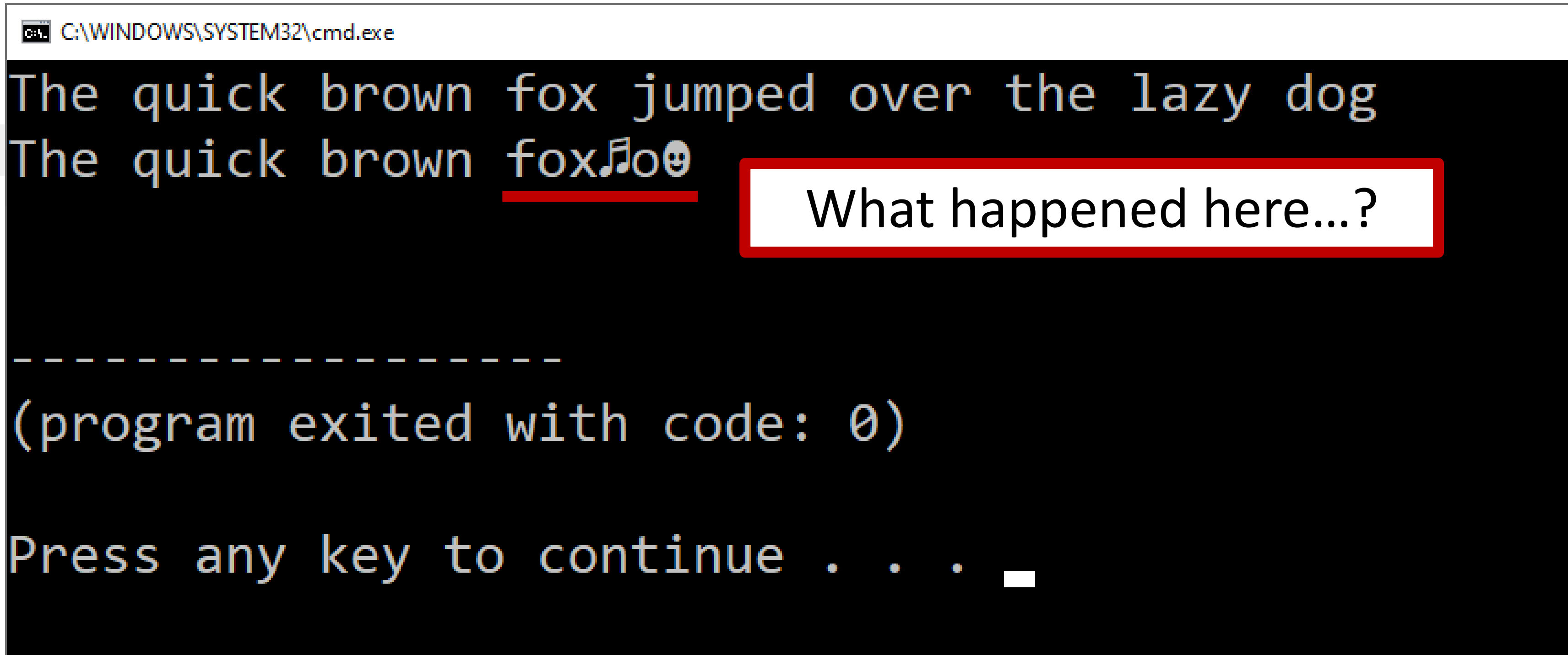
Copy first 19 characters

Print both strings



strncmp.c

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

int main (void)
{
    char str[] = "The quick brown fox jumped over the lazy dog";
    char dst[64];

    strncpy(dst, str, 19);

    puts(str);
    puts(dst);

    return (0);
}
```



C:\WINDOWS\SYSTEM32\cmd.exe

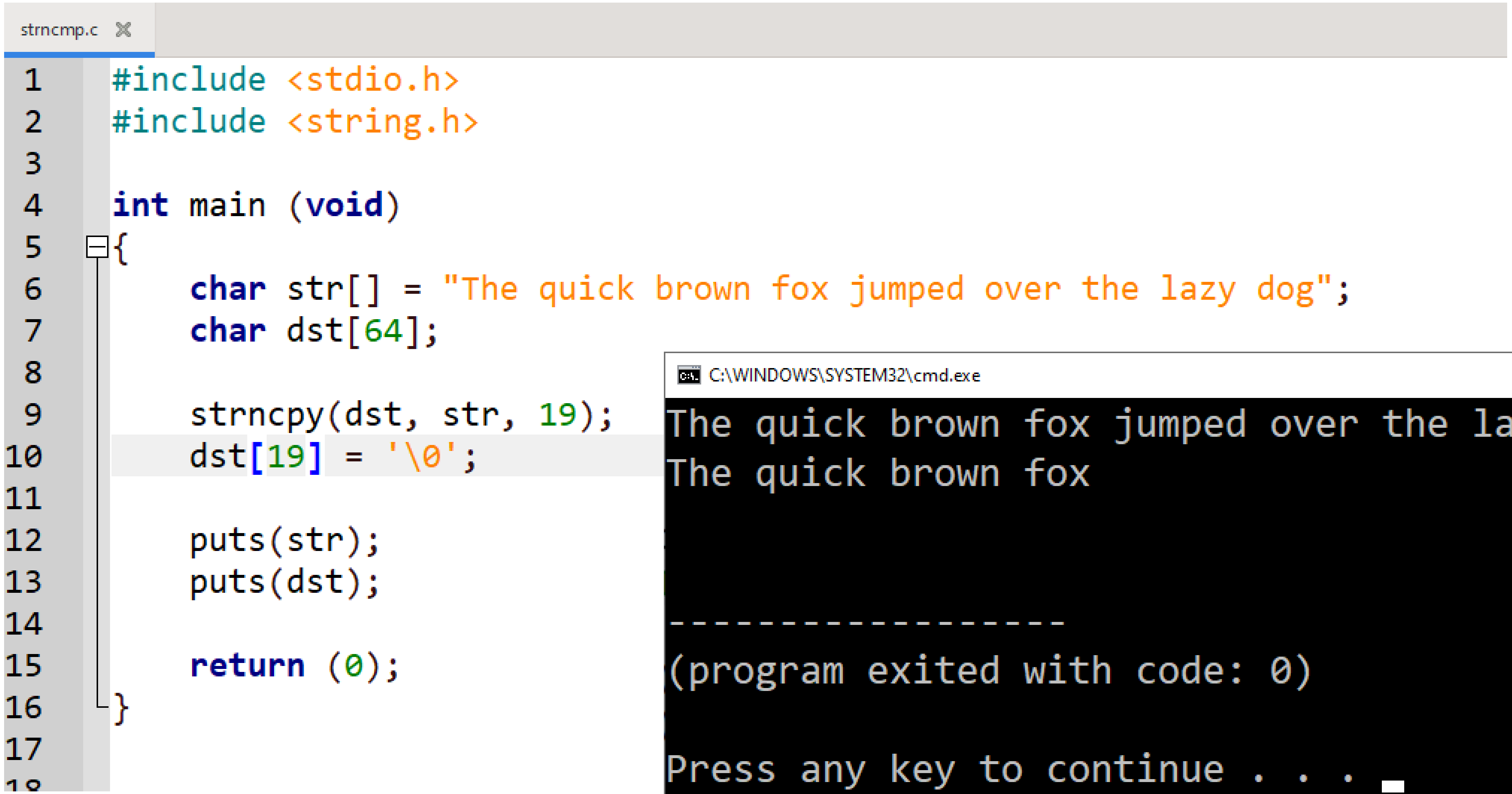
```
The quick brown fox jumped over the lazy dog
The quick brown fox♫o☻



------------------
(program exited with code: 0)

Press any key to continue . . .
```

What happened here...?

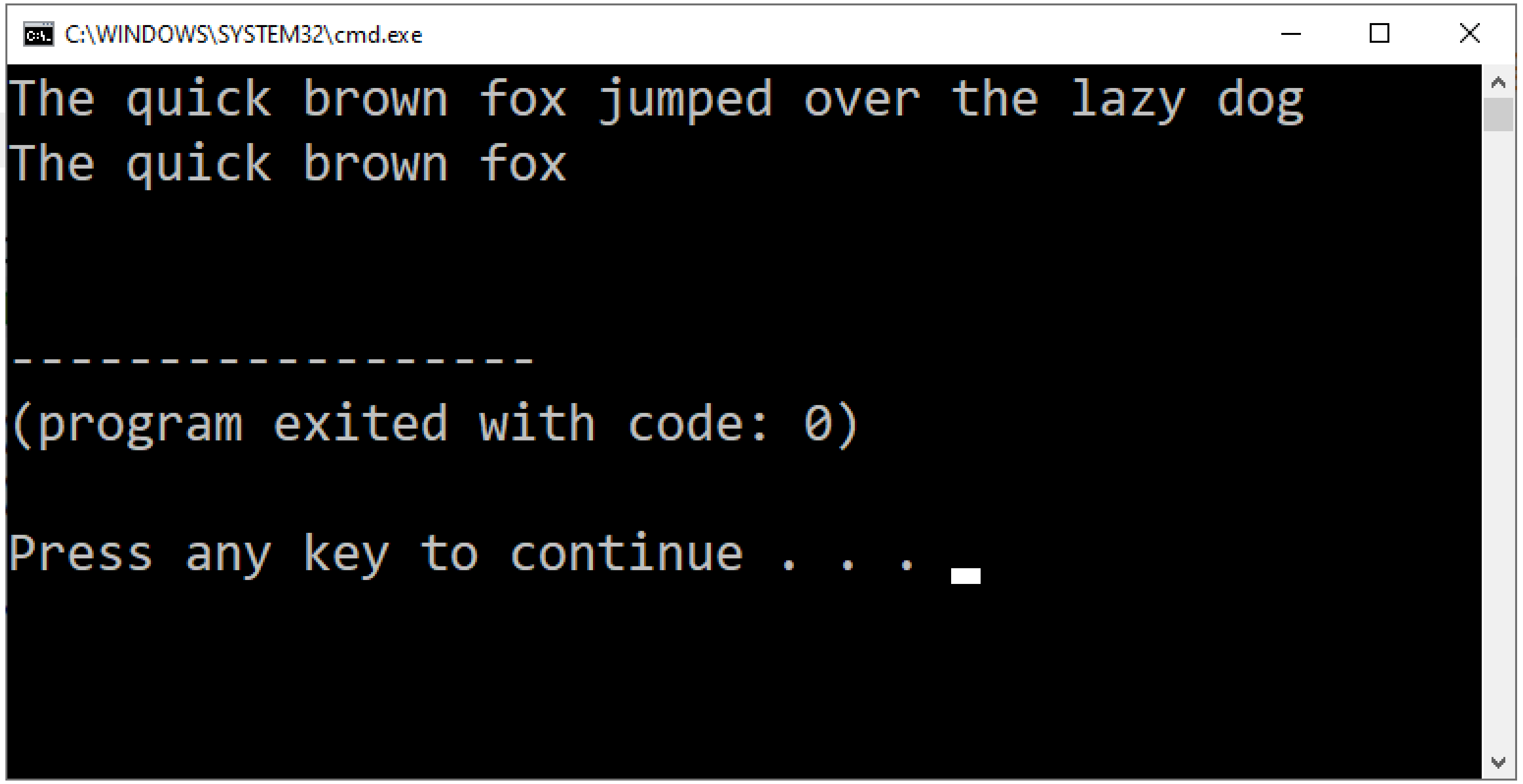
strncpy does not insert the null character!

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

int main (void)
{
  char str[] = "The quick brown fox jumped over the lazy dog";
  char dst[64];

  strncpy(dst, str, 19);
  dst[19] = ‘\0’;

  puts(str); puts(dst);

  return (0);
}
```

We must do it ourselves


strncmp.c

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

int main (void)
{
    char str[] = "The quick brown fox jumped over the lazy dog";
    char dst[64];

    strncpy(dst, str, 19);
    dst[19] = '\0';

    puts(str);
    puts(dst);

    return (0);
}
```


C:\WINDOWS\SYSTEM32\cmd.exe

```
The quick brown fox jumped over the lazy dog
The quick brown fox


--------------------
(program exited with code: 0)

Press any key to continue . . .
```

More String Functions: `strcat`

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

Concatenates (joins) two strings:

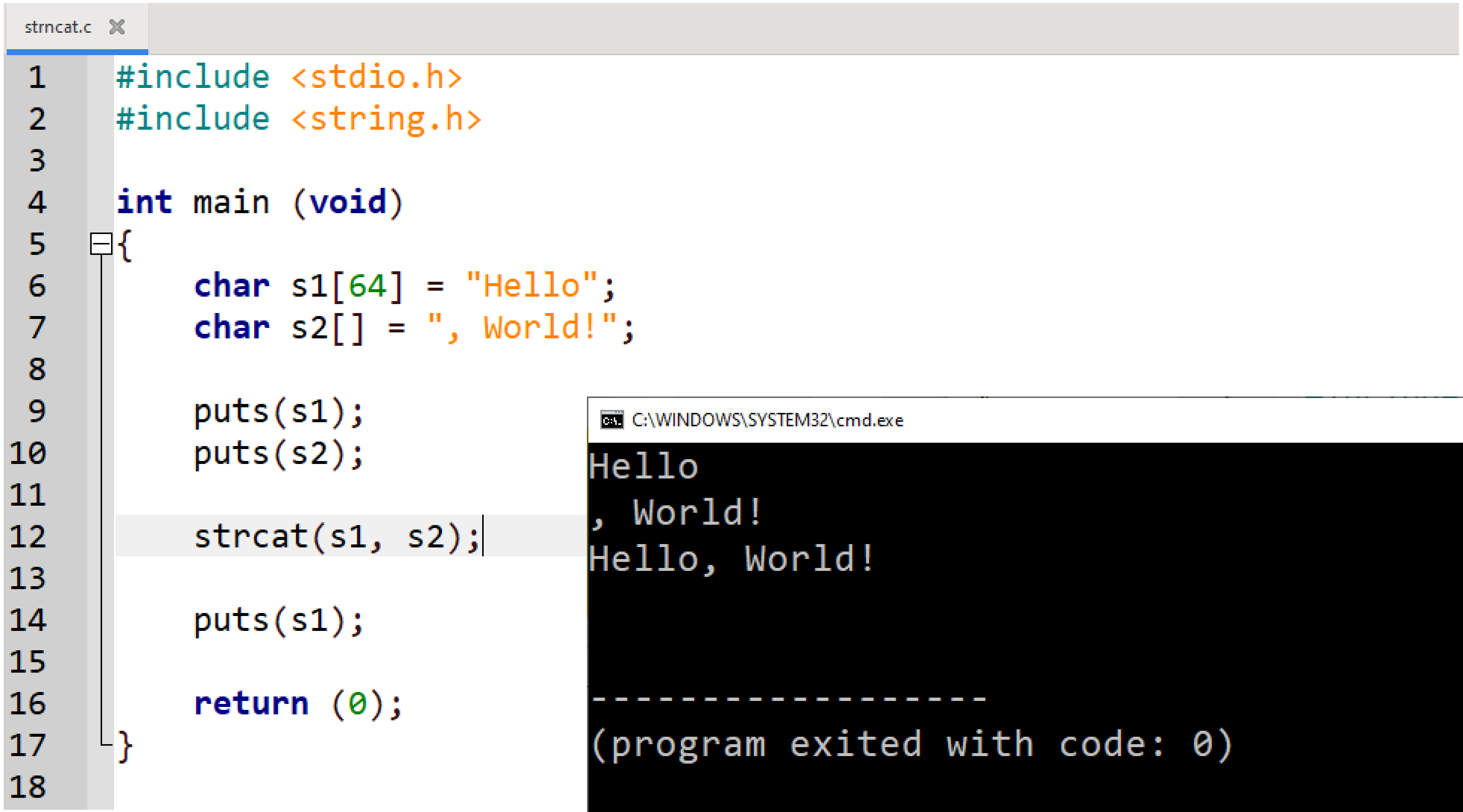
`dest`	Pointer to the destination string.
`src`	Pointer to the source string.

Appends the **`src`** string to the **`dest`** string.

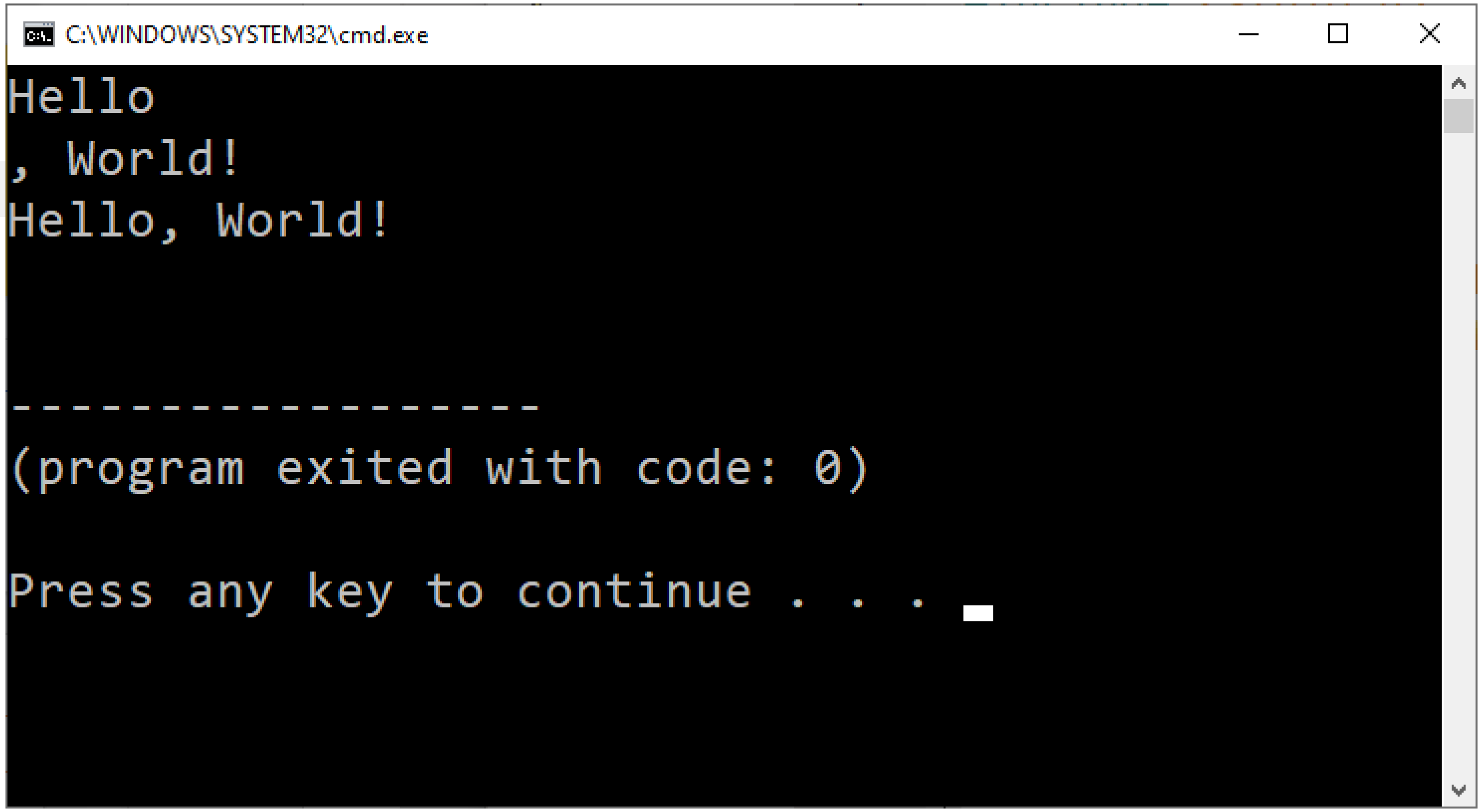
Returns a pointer to the joined string (`dest`)

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

int main (void)
{
    char s1[64] = "Hello";
    char s2[] = ", World!";

    puts(s1);
    puts(s2);

    strcat(s1, s2);

    puts(s1);

    return (0);
}
```

- Once again, make sure dest string is large enough
- We've allocated 64 characters
- Only used six (Hello + \0)



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

int main (void)
{
    char s1[64] = "Hello";
    char s2[] = ", World!";

    puts(s1);
    puts(s2);

    strcat(s1, s2);

    puts(s1);

    return (0);
}
```



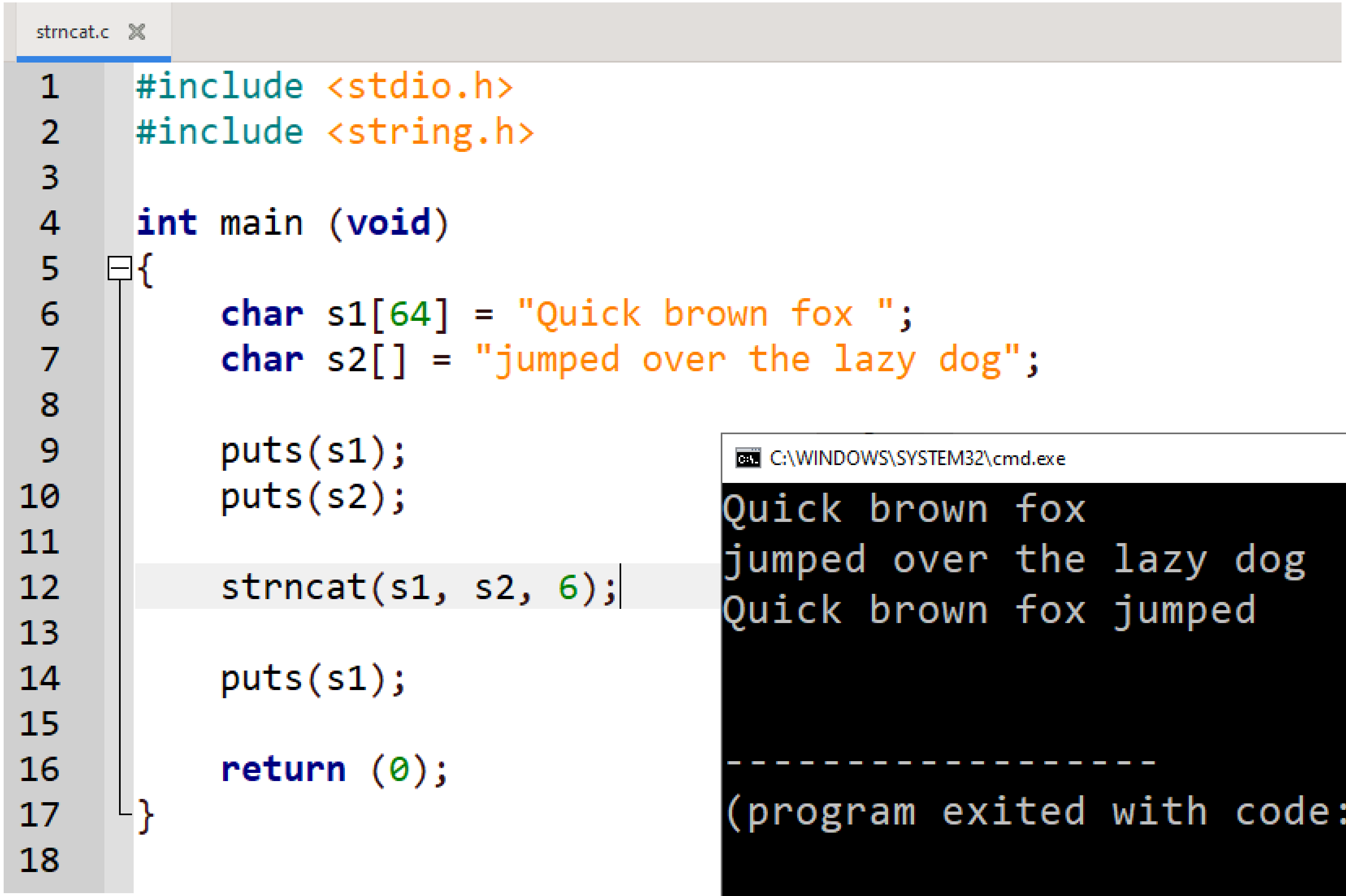
```
Hello
, World!
Hello, World!


------------------
(program exited with code: 0)

Press any key to continue . . .
```

More String Functions: `strncat`

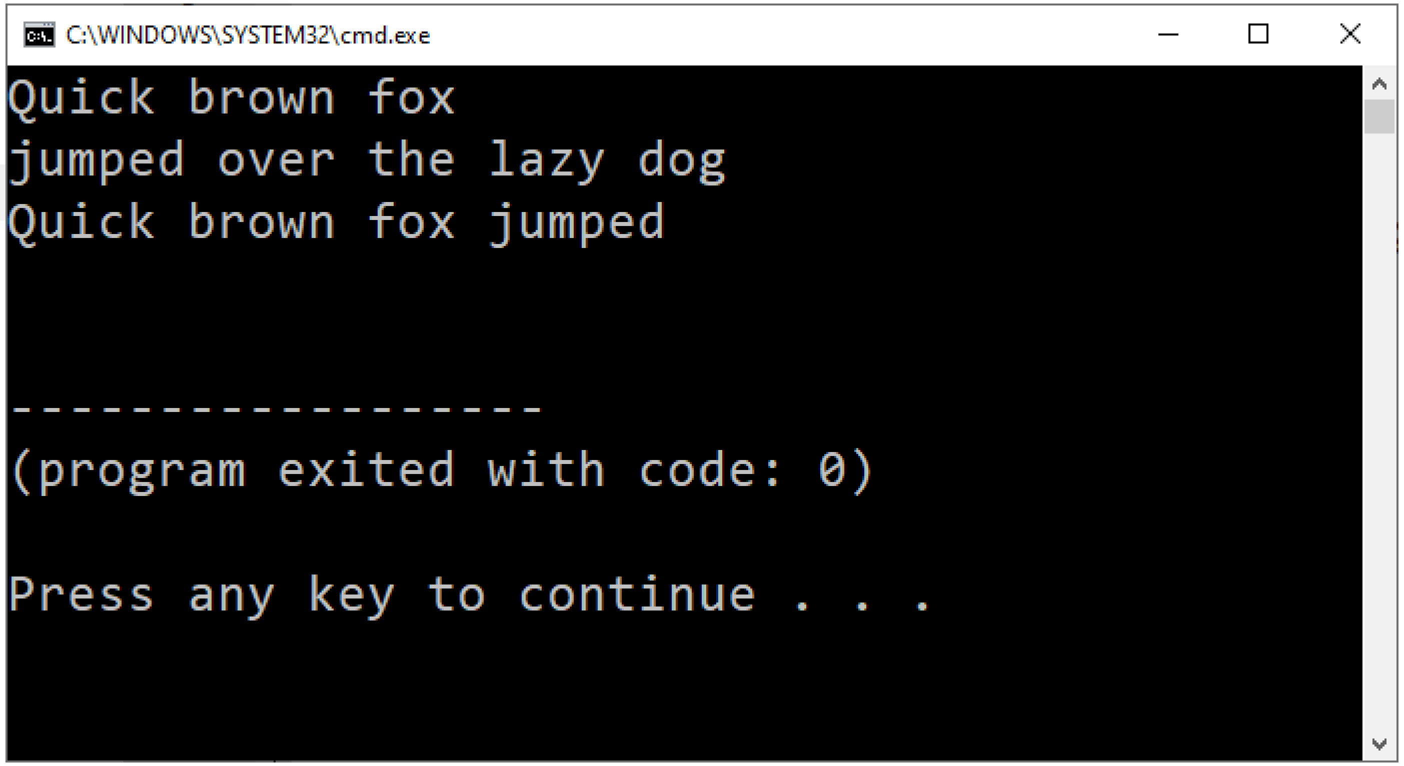
```
char *strncat(char *dest, const char *src, size_t n)
```

Appends the first **n** characters of **`src`** to **`dest`**

- **strncpy** does NOT null terminate…
- But **strncat** DOES.
- We don't have to worry about adding the null character.


strncat.c

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

int main (void)
{
    char s1[64] = "Quick brown fox ";
    char s2[] = "jumped over the lazy dog";

    puts(s1);
    puts(s2);

    strncat(s1, s2, 6);

    puts(s1);

    return (0);
}

```


C:\WINDOWS\SYSTEM32\cmd.exe

```
Quick brown fox
jumped over the lazy dog
Quick brown fox jumped


------------------
(program exited with code: 0)

Press any key to continue . . .
```

Next Class

More string functions
More string examples

Questions?