C Programming

Strings

Strings

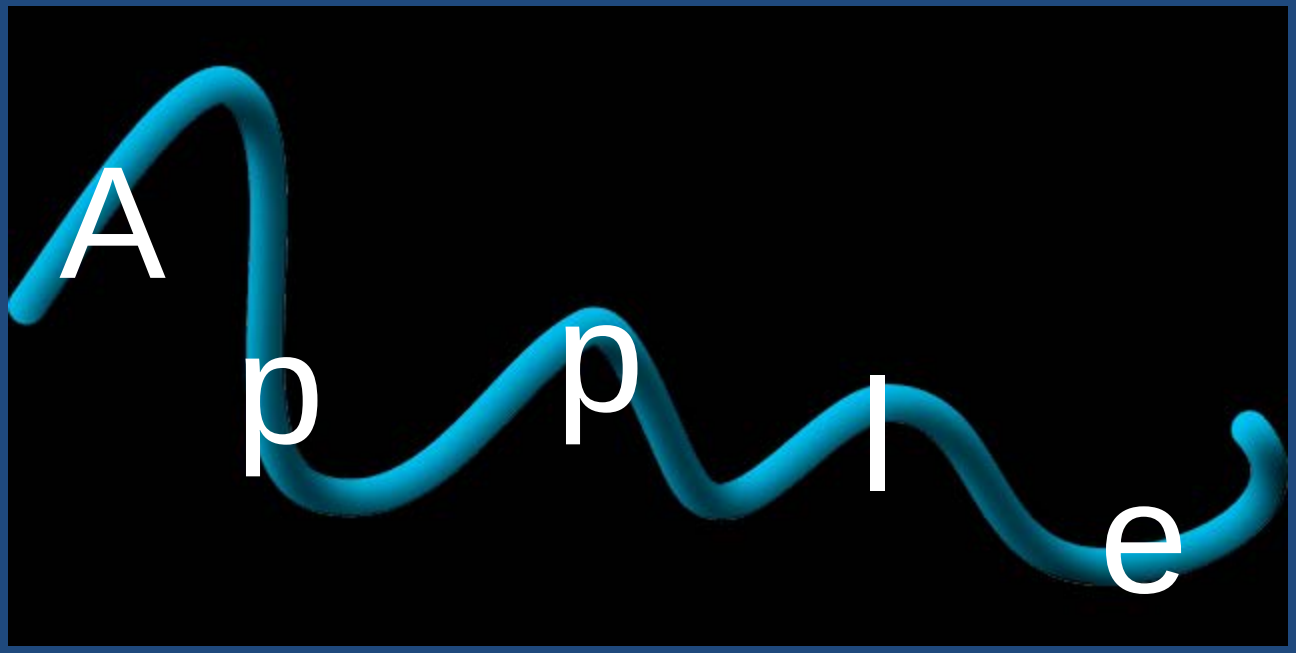
"DO OR DO NOT. THERE IS NO TRY."

"HELLO WORLD!"

"HAKUNA MATATA!"

"KHAAAAAAAANNNNNN!!!!"

A string is a sequence of characters



Aside: origin of the term likely comes from late 15th century as

"a number of objects arranged in a line"

(http://www.etymonline.com/index.php?term=string)

Example

```
char name[81] = "Fred Flintstone";
```

This example has 81 bytes associated with it.

FRED 4 bytes

1 byte for a space

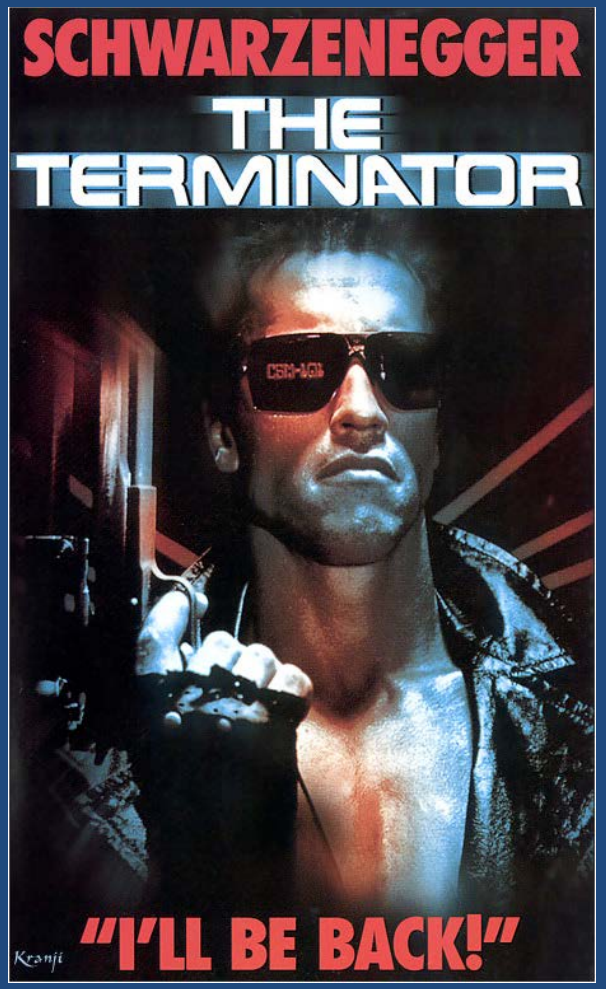
FLINTSTONE

10 bytes

plus 1 byte more!

Only
16 bytes
are used
(currently)

There's an invisible terminator



It indicates

“this is the end of the string”

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F	r	e	d		F	l	i	n	t	s	t	o	n	e	'\0'

It's the ASCII value

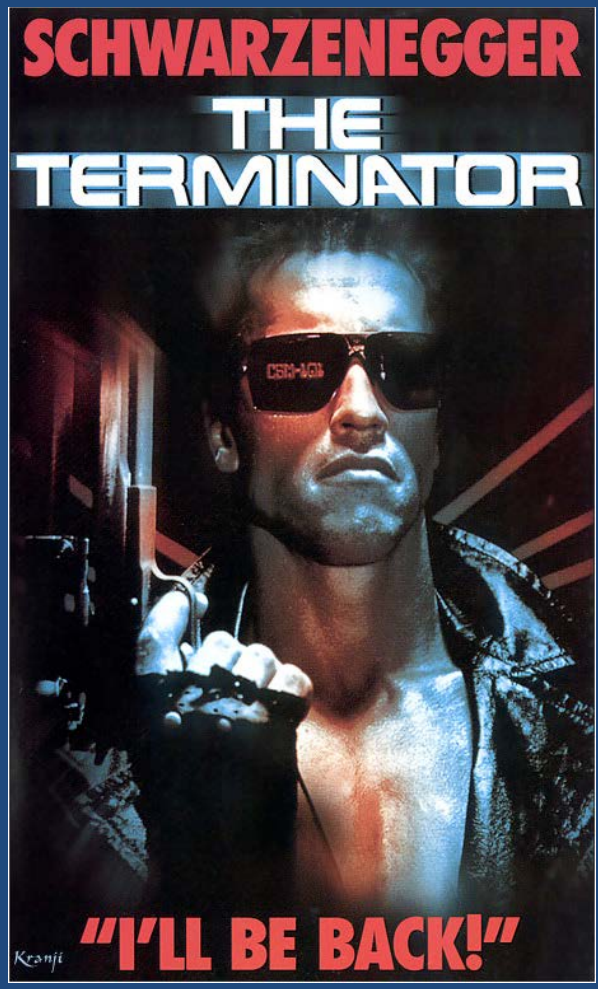
0

This is **NOT** the character '0'.

The string's initial value goes into the array, starting at element 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F	r	e	d		F	l	i	n	t	s	t	o	n	e	'\0'

Once it's done, it puts a null-termination at the end.



The null-termination is often expressed as '\0'.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F	r	e	d		F	l	i	n	t	s	t	o	n	e	'\0'

When C looks at a string variable, it starts at element 0 and keeps going until it reaches the null-termination.

Everything after the null-termination is irrelevant.

Declaring the String's Maximum Size

Allocate one **more** byte than you need for the characters you expect in the string

Overflowing the Array

Just like any other array, you don't have protection against overflowing the array

It's much easier to overflow in C than in other languages

Initializing an Empty String

You can declare a string variable that has nothing in it:

- e.g. char myName[81] = "";
- This has nothing between the two double-quotes.
- This puts a null-termination in myName[0].

Review!

If a variable is declared as

char myName[20],

what's the length of the longest name that can be stored in it?

Review!

If you were going to declare a variable to store postal codes (e.g. N2L 5R3, including the space), how big must the array be?

Review!

What's the difference between

'0',

'\0',

and 0?

Size

It is common to use a #define'd constant for the array size

```
#define kNameSize 31

char myName[kNameSize] = “”;
```

Using *const* won't work

Review!

Why should you use a constant for the array size?

What You Can't Do With Strings

Except upon the variable declaration, you can't assign a string using an =

- e.g. you can't do: myName = "Fred";
- You'll get a "left operand must be l-value" (or something like that) error.

You can't compare strings using == or any other comparison operator

- e.g. you can't do: if(myName == "Fred")

- It’s legal but wrong to do

You can't add or append strings using a plus sign

- e.g. you can't do: myName = "Fred " + "Flintstone";

NO! = NA!

==

NON! + NYET!

Use String Functions Instead

To assign a string to an array of char, use strcpy()

- e.g. strcpy(myName, "Fred");

This assumes that myName is an array of char with enough room to hold "Fred" and the null-termination.

- Not enough room?
- **It'll copy it anyway!**

To append a string to another one in an array of char, use strcat()

e.g. strcat(myName, "Fred");

Same warning as previous!

To compare two strings, use strcmp()

- e.g. if(strcmp(myName, "Fred") == 0)

VERY IMPORTANT NOTE:

strcmp() returns one of **three** values:

- 0 if the strings are equal
- a number less than 0 if the first argument is less than the second one
- a number greater than 0 otherwise

Do NOT leave out the explicit comparison

IMPLICATION: You cannot use strcmp() as if it were telling you if the arguments were equal to each other.

e.g. if(strcmp(myName, "Fred") == TRUE)

- This will give you the **wrong answer** if you're assuming that TRUE returned means that the strings are equal!

e.g. if(strcmp(myName, "Fred"))

- This will also give you the **wrong answer** if you're assuming that TRUE returned means that the strings are equal!

Comparing Strings

Strings are compared "lexicographically"

i.e. the ASCII values of the characters are numerically compared, in order, until a mismatch is found or the end of a string is found

Thus, "a" (ASCII value 97) is greater than "A" (ASCII value 65)

If you want to do a case-insensitive comparison, use stricmp().

Review!

How do you compare two strings?

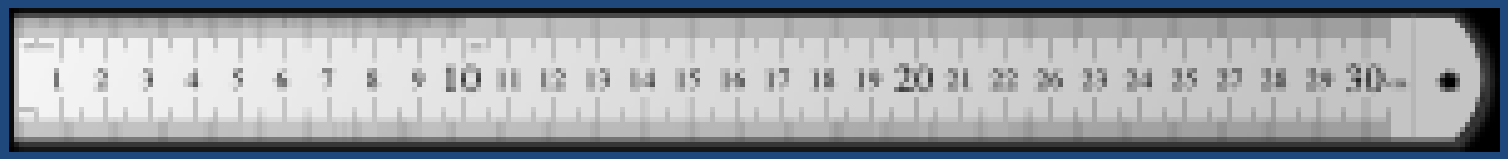
Review!

How do you copy one string to another string?

More String Functions

To find out how long a string is, use strlen()

- e.g. len = strlen(myName);



len must be of type *size_t*

- *size_t* is a data type that can contain the longest length of a string in that particular operating environment.
- On our systems, it is equivalent to unsigned int, so we can have rather long strings.

One way to convert a string to an int is to use atoi()

- e.g. value = atoi(myNumber);

This looks at *myNumber* and

returns an int representing the number found there

if there is one leading off the string (ignoring whitespace)

If there is a non-number before any numbers, it stops there and returns 0

If there is a non-number **after** any numbers, it stops there and returns the number found up to that point

(minus signs are assumed to be parts of numbers)

atoi() Conversions

String	Value returned from atoi()
1	1
140	140
140e10	140
e10	0
140.4	140
140 899	140

Other Conversion Functions

There are other functions that convert from strings to numbers:

- e.g. atof(), atol()

They behave in the same basic way, keeping in mind the differences of the data that you're keeping track of

- e.g. atof() will accept decimal points.

One very counter-intuitive thing to know:

- atof() does **NOT** return a float

It returns a double

Other String Functions

There are TONS of other string functions.

Example of String Usage

```
char str[81] = "";
char str2[81] = "Barney";
    strcpy(str, "Fred");
    printf("%s\n", str);
    strcat(str, " Flintstone");
    printf("%s\n", str);
    if( strcmp(str, str2) == 0 )
        printf("str and str2 are both %s", str);
    else
        printf("%s and %s are different\n", str, str2);
```

#include

You **must** use

#include <string.h>

If you are using atoi(), etc.,

#include <stdlib.h>

Review!

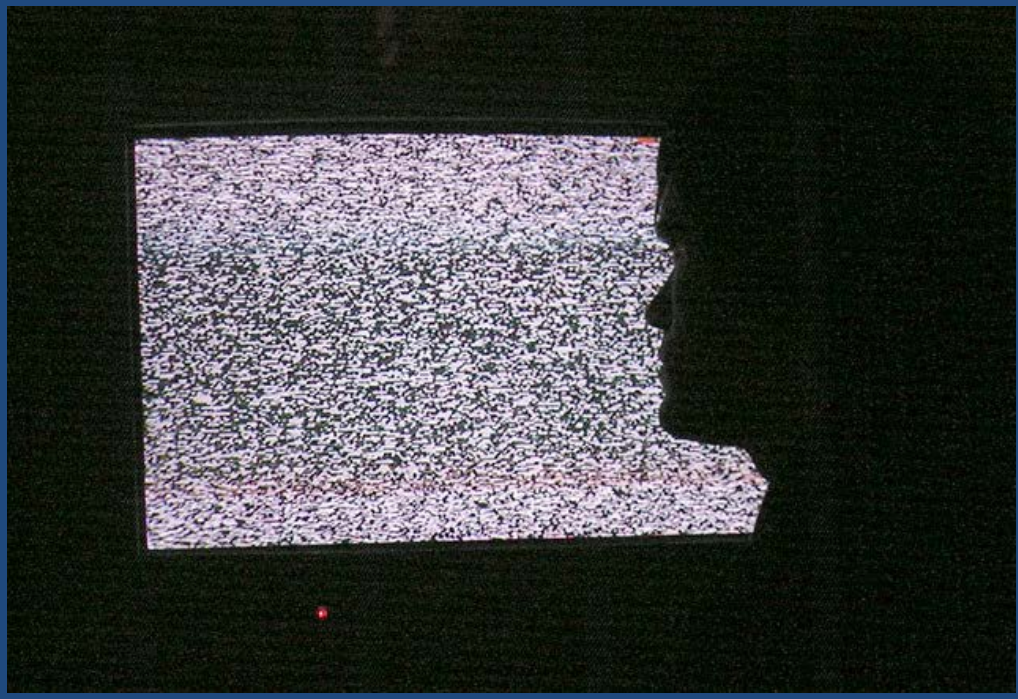
Why do you have to #include <string.h>?

On a related note ...

You want to be careful about storing large arrays on the stack

If the array is too large, it'll overflow the stack

A workaround is to use the static storage class

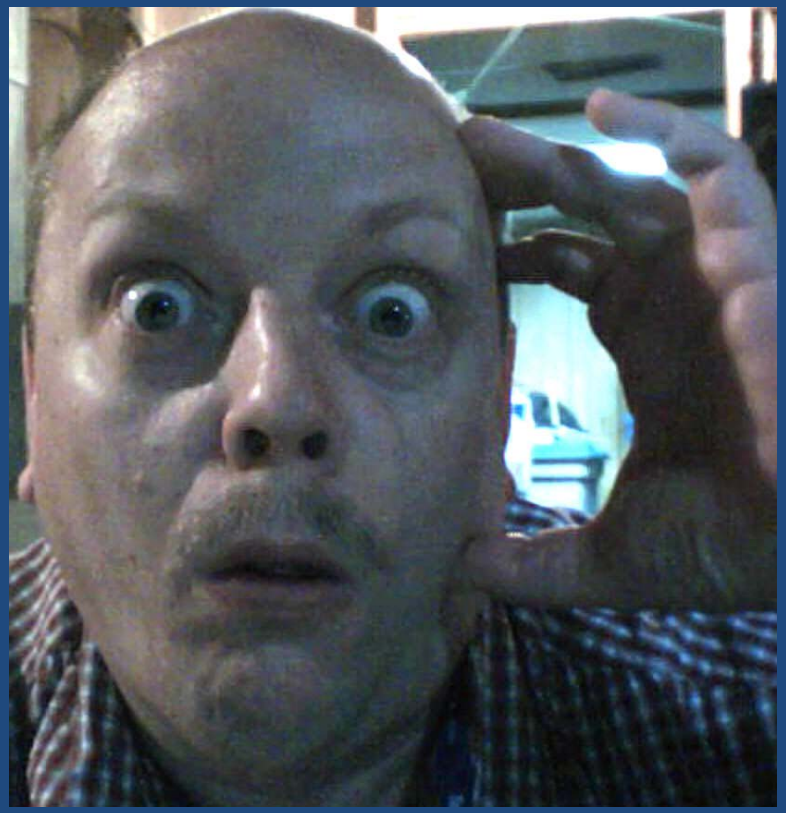


This stores the variable in a separate area called the heap

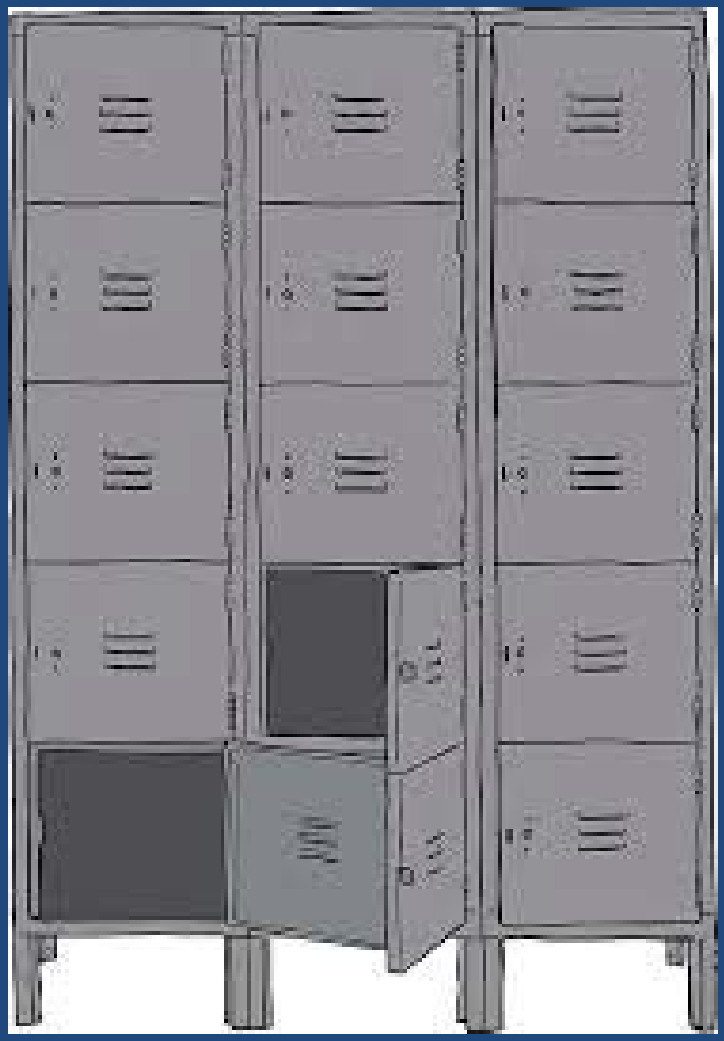
No longer on the stack

Side-effect: static variables are created when the program starts and are destroyed when the program ends

Even if they are declared in a function!



So, the value is retained from one call of the function to the next



How? Simply put static before the variable declaration:

```
static int howMany = 0;
static int lotsOfData[1000000]
    = {0};
```

Review!

What's a significant side-effect of using static?

Summary

1. Strings are null-terminated arrays of char.
2. You have to worry about overflowing the array.
3. Use string functions.