C Programming
Strings

Adam Sampson (based on material by Greg Michaelson)
School of Mathematical and Computer Sciences
Heriot-Watt University

C lectures

- Compiling code, program layout, printing/reading data, expressions, arithmetic, memory addresses, control flow, precedence
- Functions, pointers, file IO, arrays
- Memory allocation, casting, masking, shifting
- **Strings, structures, dynamic space allocation, field access**
- Recursive structures, 2D arrays, union types

Recap: arrays

- A C array is stored in a series of contiguous memory locations, numbered from `0` to *`length`*`-1`
- At runtime, the C program **does not know** the length of the array – it's not stored anywhere automatically
- Strings in C are similar...

The C string

- A string is stored in memory as an array of `chars`
- The last `char` is `'\0'` (a zero byte)
- No additional length information

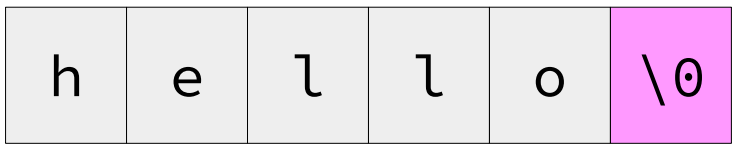
- `"characters"` is a **string literal** (or string constant)
- Allocates space for characters , plus the `'\0'`
- Value is a pointer to the first character

String variables

`char name[size];`

- Allocates space for `size` characters
- Cannot assign another string to *name*: must copy character by character

Remember to include space for the `'\0'`: `"hello"` is **6** characters!



`char *name;`

- Pointer to a string you've allocated elsewhere
- *Can* assign another string to *name* – then both will point at the same string

`const`

- The type of a string literal is **`const`** `char *`
  - (or **`const`** `char []`)
- `const` means “constant”: you aren't allowed to use this pointer to change the thing that it points at
- Can convert a non-`const` pointer to `const` automatically, but not the other way round
- Functions taking a string argument will usually take a `const char *` if they don't need to change it

`slength.c`

String length

String length

```
#include <stdio.h>

int slength(const char *s)
{
    int i = 0;
    while (s[i] != '\0')
        i++;
    return i;
}
```

String length

```
int main(int argc, char *argv[])
{
    const char *q = "how long is this string?";
    printf("%s: %d characters\n",
           q, slength(q));
    return 0;
}

$ ./slength
how long is this string?: 24 characters
```

strlen

- The C standard library includes many useful string functions, mostly defined in `<string.h>`
- `slength` is like `<string.h>`'s **`strlen`** function

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

- Returns the length of a string, **not** including the `\0`
- We saw `size_t` before – big enough to measure the size of any range of memory – `int` might not be!

String comparison

- Strings s_0 and s_1 are **equal** if...
  - Same length: n
  - For 0 <= i <= n-1: $s_0[i] == s_1[i]$
  - “banana” equals “banana”
- String s_0 is **less than** string s_1 if...
  - For 0 <= i < j: $s_0[i] == s_1[i]$
  - $s_0[j] < s_1[j]$
  - “banana” is less than “banish”
  - “ban” is less than “band”

String comparison

- String s_0 is **greater than** string s_1 if...
  - For 0 <= i < j: $s_0[i] == s_1[i]$
  - $s_0[j] > s_1[j]$
  - “banquet” is greater than “banana”
  - “bank” is greater than “ban”

`scomp.c`

String comparison

String comparison

```
#include <stdio.h>

int scomp(const char *s0, const char *s1)
{
    int i = 0;
    while (s0[i] == s1[i]) {
        if (s0[i] == '\0')
            return 0;
        i++;
    }
    if (s0[i] == '\0' || s0[i] < s1[i])
        return -1;
    return 1;
}
```

String comparison

```
int main(int argc, char *argv[])
{
    printf("banana banana %d\n",  scomp("banana", "banana"));
    printf("banana banish %d\n",  scomp("banana", "banish"));
    printf("ban band %d\n",       scomp("ban", "band"));
    printf("banquet banana %d\n", scomp("banquet", "banana"));
    printf("bank ban %d\n",       scomp("bank", "ban"));
    return 0;
}
```

```
$ ./scomp
banana banana 0
banana banish -1
ban band -1
banquet banana 1
bank ban 1
```

strcmp

- `scomp` is like `<string.h>`'s **`strcmp`** function

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

- Compares two strings, returning -1, 0 or 1
- Idiom for testing string equality (as non-zero is true) – `if (!strcmp(a, b)) { ... }`