

# Strings I

- there are no defined type called `string` in C!
- instead we deal with `char *` or char array objects
- string declaration and initialization:

```
#define LEN_SMALL 10
```

```
char *str1 = "hello", str2[LEN_SMALL];
```

- printing a string:

```
printf("str1 = %s\n", str1);
```

- strings or `char *`s in C are always saved as *null-terminated* arrays (null is the special character “\0”) i.e. internally `str1` is stored as `hello\0`
- thats the reason for a string or a `char *` its possible to know its length, for `int` or `doubles` there is no such special number to be used as end-of-array marker

## Strings II

- printing before initialization would produce garbage
- long-hand initialization (after declaration):

```
str2[0] = 't';  
str2[1] = 'h';  
str2[2] = 'e';  
str2[3] = 'r';  
str2[4] = 'e';  
str2[5] = '\0';
```

- now printing would work fine:

```
printf("str2 = %s\n", str2);
```

## Strings III

- some useful function exported by #include <string.h>

```
size_t  
strlen(const char *s);  
char *  
strcpy(char *dst, const char *src);  
char *  
strdup(const char *str);
```

- some esoteric ones (to whet your appetite):

```
char *  
index(const char *s, int c);  
char *  
rindex(const char *s, int c);  
char *  
strchr(const char *s, int c);  
char *  
strstr(const char *big, const char *little);  
char *  
strtok_r(char *str, const char *sep, char **last);
```

- there are plenty more, you should try out (bon appetite!)

## Strings IV

- a simple of the string library functions:

```
char *
str_add (char *str1, char *sep, char *str2, int max_length)
{
    char *ret_str = NULL;

    ret_str = (char *) malloc(max_length * sizeof(char));
    ret_str = strcpy(ret_str, str1);
    ret_str = strcat(ret_str, sep);
    ret_str = strcat(ret_str, str2);
    return ret_str;
}
```

- quick question: in the above function, who is responsible for memory management, the caller or the calle?
- also, can you get rid of the max\_length argument? think about strlen( )
- use of the above function:

```
printf("str1 + str2 = %s\n", str_add(str1, "", str2, LEN_BIG));
```

## Strings V

- command line arguments:

```
int  
main (int argc, char **argv)
```

- the special function has `main( )` gets `argc` many arguments (from the command line, including the program name itself) and these are stored in the `char *s` i.e. character arrays:

```
argv[0], argv[1], ..., argv[argc - 1]
```

- the following code will spit out what the program got from the command line:

```
for (ii = 0; ii < argc; ++ii)  
    printf("argv[%d]: \"%s\"\n", ii, argv[ii]);
```

## Strings VI

- if the program is called a.out then:
  - ./a.out would produce:  
argv[0]: "./a.out"
  - ./a.out -d 2 would produce:  
argv[0]: "./a.out"  
argv[1]: "-d"  
argv[2]: "2"

## Code Files

prog8.c