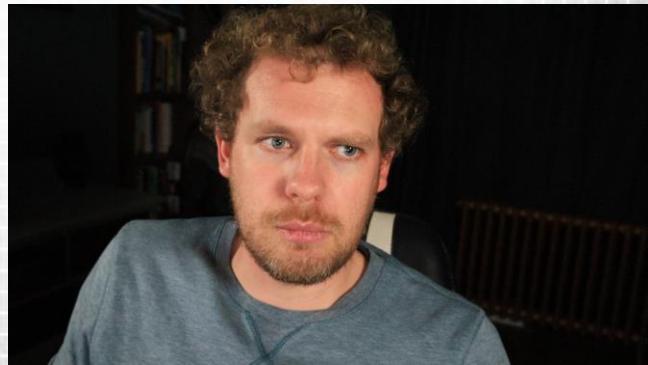
Parallel Computing with GPUs

An Introduction to C
Part 3 – Arrays Strings and IO

The University Of Sheffield.

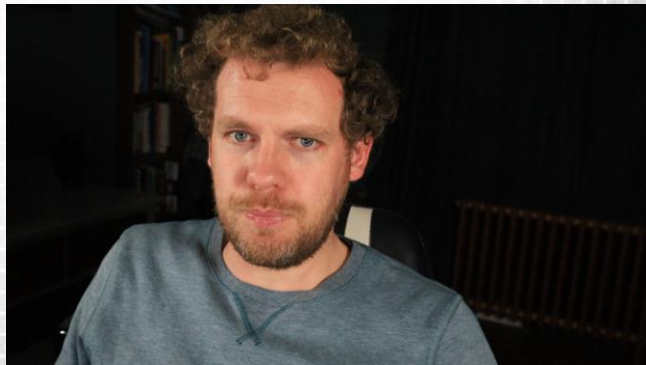
Dr Paul Richmond

http://paulrichmond.shef.ac.uk/teaching/COM4521/



This Lecture (learning objectives)

- ❑ Arrays, strings and basic IO
  - ❑ Identify definitions of arrays on the stack
  - ❑ Select appropriate *formats* to perform effective program input and output
- ❑ File IO
  - ❑ How to operate on files as streams



Arrays

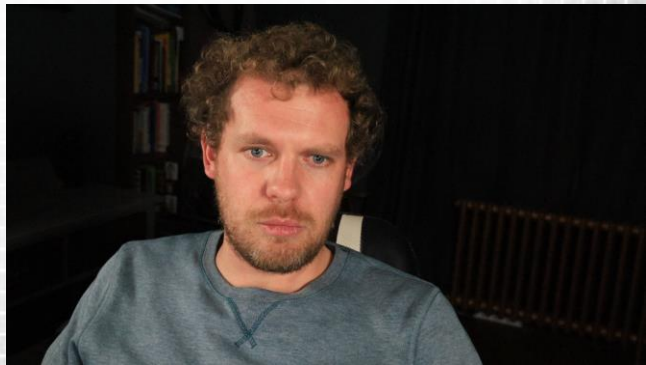
- Arrays can be compile time defined using `[size]`
  - Local arrays will be created on the stack (not heap)
  - Multidimensional Arrays possible
- Character Arrays
  - Represent strings
  - String literals can be assigned to an array at declaration only
  - Termination required with `'\0'` character
  - `char *name` is equivalent to `char name []`

```
char my_string1[] = "hello";
char my_string2[6] = "hello";
char my_string3[6] = { 'h', 'e', 'l', 'l', 'o', '\0' };
char *mystring4 = "hello";
static const char mystring5[] = "hello"; //can't be modified

char my_string6[6];
my_string5 = "hello"; //ERROR

char my_string7[5] = "hello" //ERROR
```

Heap vs. Stack

- Stack
  - Memory is managed for you
  - When a function declares a variable it is pushed onto the stack
  - When a function exits all variables on the stack are popped
  - Stack variables are therefore local
  - The stack has size limits (1Mb in VS2017)
- Heap (next lecture)
  - You must manage memory
  - No size restrictions (except available memory)
  - Accessible by any function
- Other
  - Global variables stored in a special data area of memory
  - Program stored in code area of memory

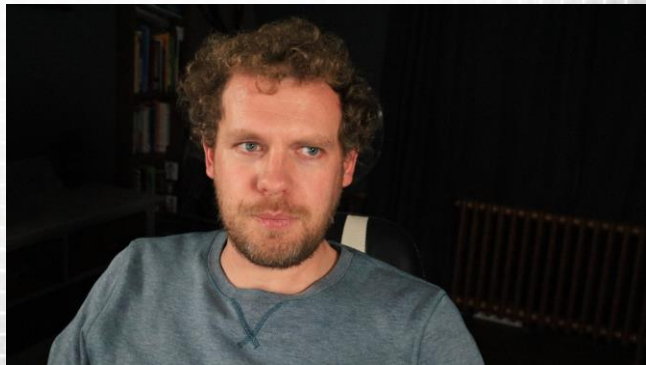
Extra Reading: http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/



Basic IO

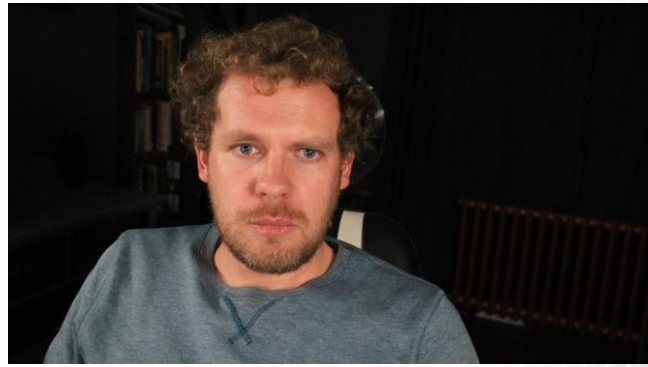
- Text Stream abstraction for all input output
  - `stdin`: Standard input
  - `stdout`: Standard output
  - `stderr`: Standard Error
  - `stdin` and `stdout` can be manipulated by;
    - `int getchar();`
    - `int putchar(int c);`

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

void main()
{
    int c;
    while ((c = getchar()) != '\n')
        putchar(toupper(c));
}
```



Formatted IO

- Output: `printf`
  - Print (to `stdout`) using formatted string
  - Format specification string and variables
- Input: `scanf`
  - Scans input (from `stdin`) according to format string
  - Saves input to variables in given format
  - Return value is the number of arguments filled
  - Variable argument are pointer to variables (`&`)
    - More on this next lecture…



```
printf("integer variable a value is %d", a);
printf("float variable b value is %f", b);
scanf("%d", &myint);
scanf("%f", &myfloat);
```

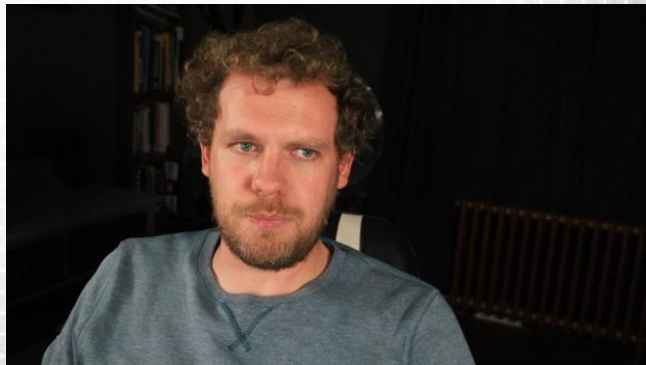
String formatting: Common format specifiers

`%[flags][width][.precision][length]specifier`

Specifier	Output	Example
`d or I (lld)`	Signed integer (long long signed integer)	123, -123
`U (llu)`	Unsigned integer (long long unsigned integer)	123
`x or X`	Unsigned hexadecimal integer (X uppercase)	1c4, 1C4
`f`	Decimal floating point	123.456
`e or E`	Scientific notation (E uppercase)	6.64e+2, 6.64E+2
`c`	character	A
`s`	Terminated string of characters	character string

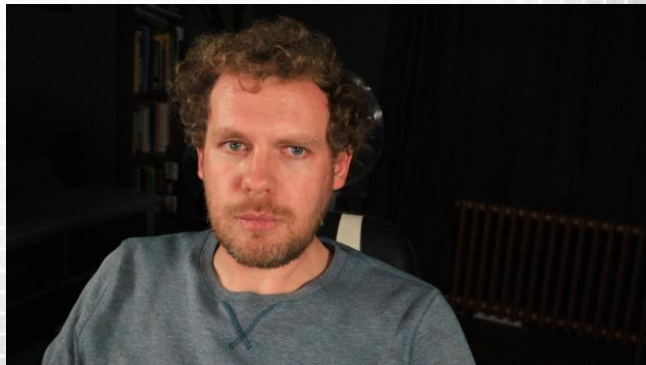
Flag	Description
–	Left justify given width
+	Forces use of + or - sign
0	Left pads the number with zeros (0)

.precision	Description
`.number`	For `d`, `u` or `i` to minimum number of digits For `f` and `e` the number of decimal places after decimal point



String Formatting Escape Characters

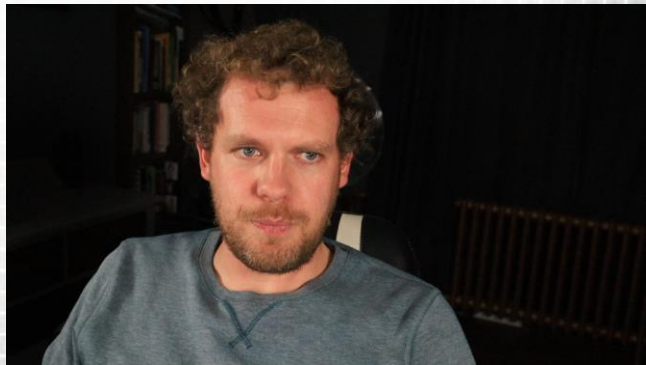
Escape Sequence	Character represented
\a	Alarm beep (system beep)
\b	Backspace
\f	Formfeed (new page), e.g. new page in terminal
\n	New line or line feed
\r	Carriage return
\t	Horizontal tab
\\	Backslash
\’ or \” or \?	Single or double quotes or question mark



Formatting examples

```
printf("\t%0.4d\n", 1);

printf("\t%0.4d\n", 12345678);

printf("%d\n", (int)1.23456);

printf("%d\n", sizeof(1.95f));
```

What will each of the following output?

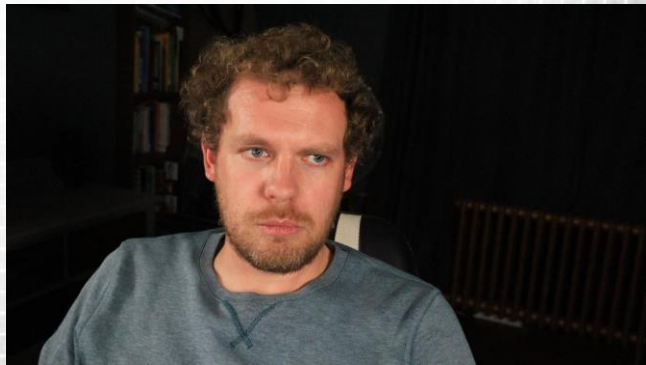


Formatting examples

```
printf("\t%0.4d\n", 1);

printf("\t%0.4d\n", 12345678);

printf("%d\n", (int)1.23456);

printf("%d\n", sizeof(1.95f));
```

```
        0001
        12345678
1
4
```

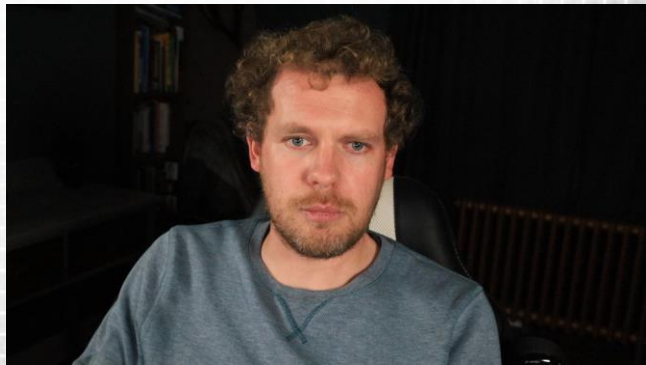
What will each of the following output?



Formatted string input and output

- `sprintf`
  - The same as `printf` but operates on a character array
- `sscanf`
  - The same as `scanf` but operates on a character array

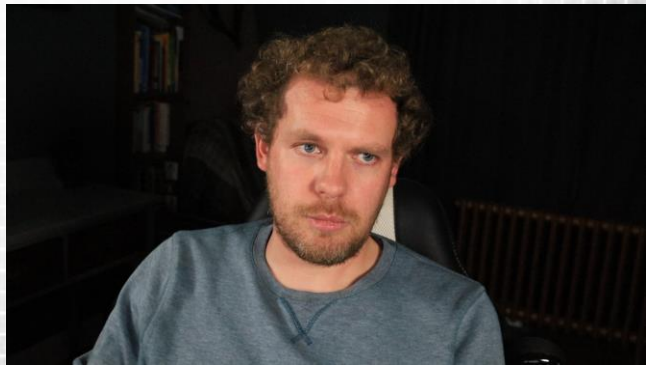
```
char s1[] = "COM4521";
int module;
char buffer[32];

sscanf(s1, "COM%d", &module);
sprintf(buffer, "COM%d is awesome!", module);
```



IO example

- A basic calculator for summing inputs

```c
#include <stdio.h>

int main()
{
    int a, sum;
    sum = 0;

    while (scanf("%d", &a) == 1)
        printf ("\tsum:%0.8d\n", sum += a);

    return 0;
}
```

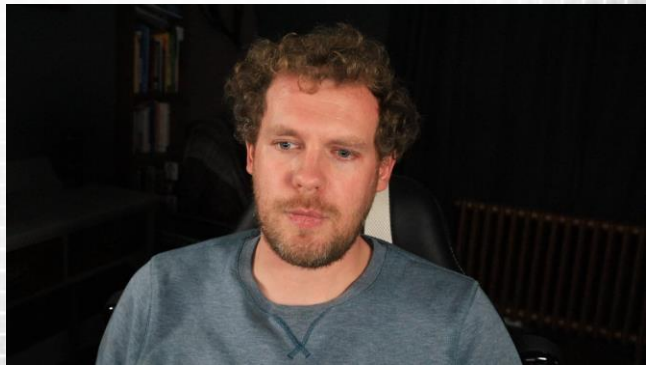


Files

- Files are still a stream
  - `FILE* fopen(char *name, char *mode);`
    - Mode: “r” = read, “w” = write, “a” = append, “b” = binary, “+” = open for update
  - `int fclose(FILE *file);`

Ignore this (*) operator for now….

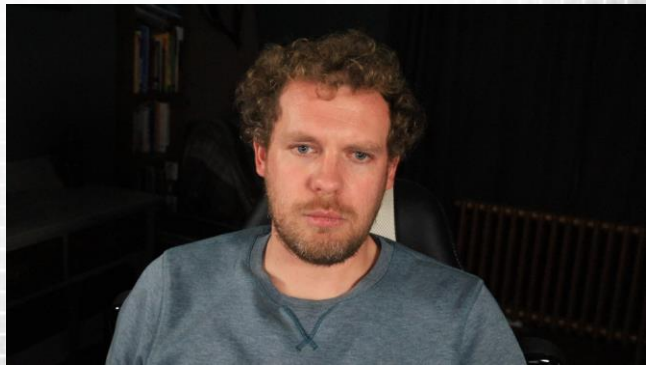
```c
#include <stdio.h>
#include <string.h>

void main()
{
    FILE *f = NULL;
    f = fopen("myfile.txt", "r");

    if (f == NULL){
        fprintf(stderr, "Could not open file\n");
    } else {
        fclose(f);
    }
}
```



File reading and writing of strings

- By character
  - `int getc(FILE *file);` same as `getchar` but on a file stream
  - `int putc(int c, FILE * file);` same as `putchar` but on file stream
- By formatted lines
  - `int fscanf(FILE *f, char *format, …);`
  - `int fprintf(FILE *f, char *format, …);`

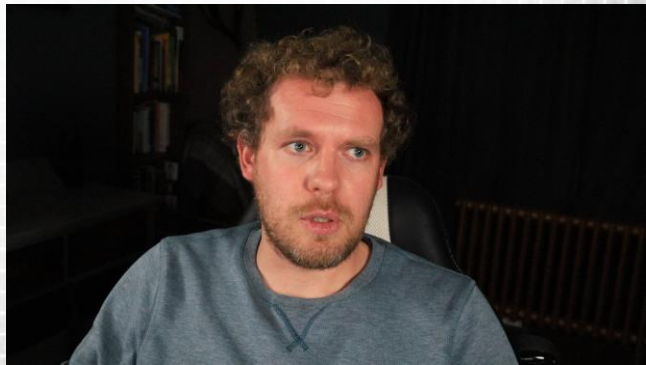
```
void filecopy(FILE* f1, FILE *f2)
{
    int c;
    while (c = getc(f1) != EOF)
        putc(c, f2);
}
```



String Coversions

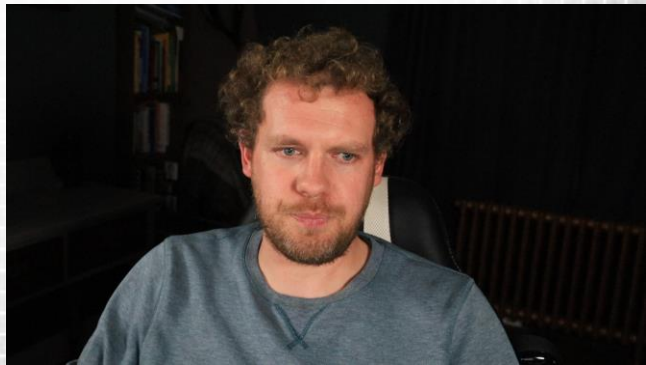
- `#include <stdlib>`
- `atof`: convert to float
- `atoi`: convert to int
- `strtod`: convert to double
- `strtoul`: convert to unsigned long

```
char *x = "450";

int result = atoi(x);

printf("integer value of the string is %d\n", result);
```



This Lecture (learning objectives)

- Arrays, strings and basic IO
  - Identify definitions of arrays on the stack
  - Select appropriate *formats* to perform effective program input and output
- File IO
  - How to operate on files as streams



Character array operations

```
char str1[20];
char str2[20];
strcpy(str1, "To be ");
strcpy(str2, "or not to be");
strncat(str1, str2, 6);
```

- `#include <string.h>`
- Copying
  - `char * strcpy ( char * destination, const char * source );`
- Compare
  - `int strcmp ( const char * str1, const char * str2 );`
  - Returns 0 if equal
- Concatenate
  - `char * strcat ( char * destination, const char * source );`
- Length
  - `size_t strlen ( const char * str );`
  - `size_t` is an unsigned integer of at least 16 bits
- `n` versions
  - Each function has a version which performs the operation up to `num` char
  - E.g. `strncpy`, `strncmp`, `strncat` all take an extra argument (…`size`