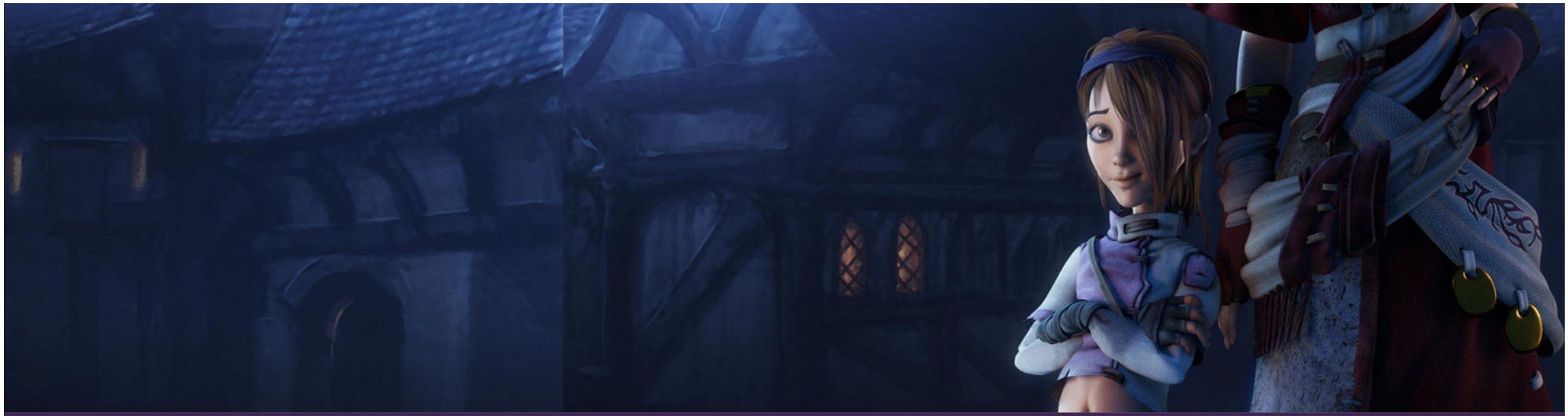

FALMOUTH
UNIVERSITY

Programming Workshop 3:
C++ for Interviews and beyond

GAM340: Professional Practice
BA(Hons) Game Development

FALMOUTH UNIVERSITY

Digital Attendance

- The system replaces paper-based registration with a network of card readers in Learning & Teaching spaces around the college.
- We hope it will save lots of time!
- You 'TAP' your ID card to register your attendance in a session.
- The window for registration is from 15 minutes before a session is timetabled to start through to 15 minutes after the start time.

- Learning Objectives
  - **Understand** the key features of C
  - **Understand** the motivations behind C++
  - **Write** working code using core C language features and libraries
  - **Understand** pointers in C
  - **Solve** basic C programming interview problems

- C & C++
  - C was first developed in 1972 as a general purpose language for running utility applications in Unix
    - Utility applications
      - Read input (user / files / punched cards)
      - Process data in memory
      - Write output (tty, screen, files, punched cards)
    - A portable language
      - Would work on different processors / architectures
      - Support for 8/16/32 bit data & address types

- C & C++
  - C++ was first developed in 1979 as ‘C with Classes’
    - 4 languages in one framework
      - C programming language
      - Classes for C
      - Template metaprogramming
      - 1993 STL

- C & C++
  - C++ was first developed in 1979 as ‘C with Classes’
    - 4 languages in one framework
      - C programming language
        - » All the goodness of C
      - Classes for C
        - » Kind of ‘object oriented’ but more ‘with objects’
        - » C# /Java are object oriented in the sense that everything is an object. This is not the case with C++
      - Template metaprogramming
        - » This is broadly equivalent to templates in C#
      - 1993 STL
        - » Templated data types (array, lists, trees, strings etc) and algorithms to operate on them
        - » This is broadly equivalent to generics in C#

- Fundamental C
  - Example_01
    - https://onlinegdb.com/HJCzgjEuH
      - Datatypes
      - Signed / unsigned data size issues
      - Control structures (for, if, do, while, switch/case)
      - rand()

- Fundamental C
  - Example_02
    - https://onlinegdb.com/HkdBXjEdS
      - Arrays
      - Address of a variable
      - A variable that stores an address
      - Accessing the value of an address
      - Iterating through memory

- Fundamental C
  - Example_02
    - Pointers, why do we do it?
      - Good question, in the olden days processors were very slow, c1-10Mhz
      - Accessing data through array lookups a[i] generates more machine code than doing pointer-based access
      - Make better use of the D-cache
      - Therefore, it's faster
      - However,
        - » Leads to lots of programming issues (run-time errors)
        - » Modern languages tend to hide raw memory from programmers

- Fundamental C
  - string.h : https://onlinegdb.com/SkbCniEdr
    - C's string library is the root of many programming tests
      - Small amount of functionality
      - All pointer-based
      - Easy to see results
      - Links into other fundamental C functionality (ctype.h)
        - » Character types (islower, isalpha etc)

- Fundamental C
  - string.h

```
char* myString = "my name is adam";  myString: "my name is adam"
```

```
0x00000001098d4f70   83c4205d  c390ff25  94000000  4c8d1d7d  |·· ]···%····L··}|
0x00000001098d4f80   00000041  53ff257d  00000090  68000000  |···AS·%}····h···|
0x00000001098d4f90   00e9e6ff  ffff6d79  206e616d  65206973  |·····my name is |
0x00000001098d4fa0   20616461  6d004865  6c6c6f2c  20576f72  | adam·Hello, Wor|
0x00000001098d4fb0   6c64210a  00000000  01000000  1c000000  |ld!·············|
0x00000001098d4fc0   00000000  1c000000  00000000  1c000000  |················|
0x00000001098d4fd0   02000000  400f0000  34000000  34000000  |····@···4···4···|
0x00000001098d4fe0   760f0000  00000000  34000000  03000000  |v······4······|
0x00000001098d4ff0   0c000100  10000100  00000000  00000001  |················|
```

- All C strings have a NULL terminator
  - This is key to make them work

- Fundamental C
  - string.h

```
char* myString = "my name is adam";  myString: "my name is adam"
```

    - NB
      - Strings that are declared like this are stored in read only memory, so you can't change them

- Fundamental C
  - string.h (*http://www.cplusplus.com/reference/cstring/?kw=string.h*)

```
char* myString = "my name is adam";  myString: "my name is adam"
```

    - NB
      - Strings that are declared like this are stored in read only memory, so you can't change them

```
char* myString = "my name is adam";  myString: "my name is adam"

char *p = myString;  p: "my name is adam"

while(*p != NULL)
{
    *p = 'a';
    p++;
}
```

- Fundamental C
  - string.h (*http://www.cplusplus.com/reference/cstring/?kw=string.h*)

```
char* myString = "my name is adam";  myString: "my name is adam"
```

    - NB
      - Strings that are declared like this are stored in read only memory, so you can't change them
        - » Also means you can't process literals in functions

```
convert_string_to_as( str: "my name is adam");
```

      - However, this works fine

```
char myString[] = "my name is adam";
```

- Fundamental C
  - string.h
    - Alternatively, let's copy the string and work with the copy

```
char* myString = "my name is adam";

char myOtherString[255];

strcpy(myOtherString,myString);

char *p = myOtherString;

while(*p != NULL)
{
    *p = 'a';
    p++;
}

printf("%s\n", myOtherString);
```

- Fundamental C
  - string.h
    - Making arbitrary strings is a bad idea, we can address that with dynamic memory allocation

```
char* myString = "my name is adam";

char* myOtherString = malloc( size: sizeof(char) * (strlen(myString)+1) );

strcpy(myOtherString,myString);

char *p = myOtherString;

while(*p != NULL)
{
    *p = 'a';
    p++;
}

printf("%s\n", myOtherString);

free(myOtherString);
```

- Fundamental C
  - string.h
    - Making arbitrary strings is a bad idea, we can address that with dynamic memory allocation
    - *malloc*
      - Will allocate bytes and return a pointer to them
    - *free*
      - Will return bytes allocated

- Fundamental C
  - Programming tests with C/C++
    - Generally looking at
      - fundamental algorithms & control flow
      - Pointer manipulation
      - Some understanding of core C library functionality
    - https://onlinegdb.com/ByFME2Vdr