demo

September 9, 2020

1 Hello World

[1]:
```
#include <iostream>

std::cout << "Hello World" << std::endl;
```

```
Hello World
```

2 Global and Local Variables

[2]:
```
// global variable
int g1 = 1;
```

[3]:
```
// local variable
{
    int l1 = 2;
}
```

[4]:
```
std::cout << l1 << std::endl;
```

```
input_line_11:2:15: error: use of undeclared identifier

'l1'
 std::cout << l1 << std::endl;
              ^

input_line_11:2:15: error: use of undeclared

identifier 'l1'
 std::cout << l1 << std::endl;
              ^



        Interpreter Error:
```

[5]:
```
std::cout << g1 << std::endl;
{
    // hide global variable
    int g1 = 3;
    std::cout << g1 << std::endl;
}
std::cout << g1 << std::endl;
```

```
1
3
1
```

3 Including and Linking

[6]:
```
%%file foo.hpp
#pragma once

namespace foo {
    int bar();
}
```

```
Writing foo.hpp
```

[7]:
```
%%file foo.cpp
#include "foo.hpp"

int foo::bar() { return 42; }
```

```
Writing foo.cpp
```

[8]:
```
!gcc -shared foo.cpp -o foo.so
```

[9]:
```
#include "foo.hpp"
```

[10]:
```
#pragma cling(load "foo.so")
```

[11]:
```
foo::bar()
```

[11]:
```
42
```

4 CUDA

[12]:
```
template <int A, int B>
class CUDA {
    int host;
    int *device;
    public:
    static __global__ void kernel(int *out){
        *out = A + B;
    }

    CUDA(){
        cudaMalloc((void**)&device, sizeof(int));
    }

    ~CUDA(){
        cudaFree(device);
    }

    int compute(){
        kernel<<<1,1>>>(device);
        cudaMemcpy(&host, device, sizeof(int), cudaMemcpyDeviceToHost);
        return host;
    }
};
```

[13]:
```
CUDA<19,23> c;
```

[14]:
```
c.compute()
```

[14]:
```
42
```

[ ]:
```
```