Week 4

Exercise 28

Note that all of the operator functions are not implemented here, as the point of this code is to show virtual inheritance.

../28/handler/handler.h

```
#ifndef INCLUDED_HANDLER_
#define INCLUDED_HANDLER_

#include "../msg/msg.h"

class Handler: public virtual Msg
{
  public:
    Handler();

  private:
};

#endif
```

../28/msg/msg.h

```
#ifndef INCLUDED_MSG_
#define INCLUDED_MSG_

#include <cstddef>

class Msg
{
  public:
    enum message
    {
      NONE    = 0,
      DEBUG   = 1 << 0,
      INFO    = 1 << 1,
      NOTICE  = 1 << 2,
      WARNING = 1 << 3,
      ERR     = 1 << 4,
      CRIT    = 1 << 5,
      ALERT   = 1 << 6,
      EMERG   = 1 << 7,
      ALL     = (1 << 8) - 1
    };

    size_t valueOf(message theEnum);
    void show(message theEnum);
};
#endif
```

../28/msg/msg.ih

```
#include "msg.h"

#include <iostream>

using namespace std;
```

../28/msg/show.cc

```
#include "msg.ih"

```

```cpp
namespace
{
  char const *name[] =
  {
    "DEBUG",
    "INFO",
    "NOTICE",
    "WARNING",
    "ERR",
    "CRIT",
    "ALERT",
    "EMERG",
  };
}

void Msg::show(message theEnum) // Taken from the solutions of week 1
{
  if (theEnum == Msg::NONE)
  {
    cout << "NONE\n";
    return;
  }

  for (size_t test = valueOf(Msg::DEBUG),                // iterates over Msg
         endTest = valueOf(Msg::EMERG) << 1,
         msgValue = valueOf(theEnum),
         idx = 0;                                  // for name[idx]

         test != endTest;
         test <<= 1, ++idx
       )
  {
    if ((test & msgValue))
      cout << name[idx] << ' ';
  }

  cout << '\n';
}
```

../28/msg/valueof.cc

```cpp
#include "msg.ih"

size_t Msg::valueOf(message theEnum)
{
  return static_cast<size_t>(theEnum);
}
```

../28/processor/processor.h

```cpp
#ifndef INCLUDED_PROCESSOR_
#define INCLUDED_PROCESSOR_

#include "../msg/msg.h"

class Processor: public virtual Msg
{
  public:
    Processor();

  private:
};

#endif
```

../28/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

#include "handler/handler.h"
#include "processor/processor.h"

#include <iostream>                // Just for testing

using namespace std;
```

../28/main.cc

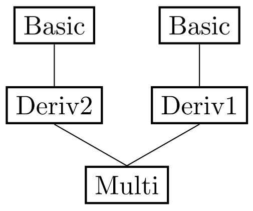
```
#include "main.ih"

int main(int argc, char const **argv) // This is just for testing
{
  Processor newProcessor;
  cout << newProcessor.valueOf(Processor::DEBUG) << '\n';
  Handler newHandler;
  cout << newHandler.valueOf(Handler::NONE) << '\n';
  newHandler.show(Msg::NONE);
}
```

Exercise 30

- Draw Multi's class hierarchy

Below is the class hierarchy of Multi at this point of the assignment.



- Explain the compiler's error message after the addition of the static_cast

As can be seen from the illustration above, due to the way that Deriv1 and Deriv2 are constructed, Basic is included twice by the time Multi inherits from Deriv1 and Deriv2. Hence, the compiler indicates that it does not know which Basic to cast to.

- Change the statement so that there is no compilation error

First, the cast can be done in a two-step fashion: first to either Deriv1 or Deriv2, and then to its associated Basic. By doing so, the compiler knows which Basic parent is applicable. This is achieved as follows:

Listing 1: c_multi.cc

```
Multi::Multi()
{
  cout << static_cast<Basic *>(static_cast<Deriv1 *>(this)) << '\n';
}
```

Secondly, a reinterpret_cast can be used instead, as follows. What this does is blindly interprets the Multi pointer (this) as a Basic pointer. Note that this is a very dangerous practice, and should be used with extreme caution, as there is a myriad of problems that can arise from this.

Listing 2: c_multi.cc

```
Multi::Multi()
{
  cout << reinterpret_cast<Basic *>(this) << '\n';
}
```

- Show the required modifications to allow the compiler to compile the statement without errors

The best way to solve the compilation error without altering the statement would be to make use of virtual inheritance. As such, the class interface of `Deriv1` and `Deriv2` should be changed as follows:

Listing 3: deriv1.h

```
...
class Deriv1: public virtual Basic
{
};
...
```

Listing 4: deriv2.h

```
...
class Deriv2: public virtual Basic
{
};
...
```

- How do you realize that this 2nd constructor is the only Basic constructor that's called

Without specifying otherwise, `Multi` directly calls the default constructor of `Basic`. To change this behaviour, explicitly calling the integer constructor in the initialisation list of `Multi`, as shown below, is the correct approach.

Listing 5: c_multi.cc

```
#include "multi.ih"

Multi::Multi()
:
  Basic(10)
{
  cout << static_cast<Basic *>(this) << '\n';
}
```