MyLogger

构造一个测试对象，在关心的函数中打 log, 从而更好理解对象的布局、生命周期管理、以及运算符的重载决策等

```
#include <cinttypes>
#include <iostream>
#include <functional>

using namespace std;

#define PRINTFUN_ADDR std::cout << static_cast<const char *>(__func__) << " at "
<< this << std::endl

class A{
    int m_a = 1;
    int m_b = 2;
    int m_c = 3;
    uint8_t m_d = 4;
public:
    A(int a, int b, int c, uint8_t d);
    A(const A& other);
    A();
    ~A();
    void Dump();
};

void A::Dump()
{
     using namespace std;
     auto dumpItem = [this](auto name, auto& item){
        std::cout << name << "@" << (void*)&item << ", size is " << sizeof(item)
<< endl;
     };
     dumpItem("m_a", m_a);
     dumpItem("m_b", m_b);
     dumpItem("m_c", m_c);
     dumpItem("m_d", m_d);
     cout << "The object size is " << sizeof(*this) << endl;
}

A::A()
{

}

A::A(int a, int b, int c, uint8_t d):
m_a(a),
m_b(b),
m_c(c)
{
    PRINTFUN_ADDR;
```

```
}

A::A(const A& other):
m_a(other.m_a),
m_b(other.m_b),
m_c(other.m_c),
m_d(other.m_d)
{
    PRINTFUN_ADDR;
}

A::~A()
{
    PRINTFUN_ADDR;
}

int main()
{
    A a{};
    A b{3,1,4,5};
    a.Dump();
    b.Dump();
}
```

工业级的 logger
[logging.h in tensorflow]

理解内存布局和对齐

在上节的测试类中增删改各种成员变量，并尝试将对象作为函数参数调用和返回，利用打印输出观察它们对大小和对齐的影响

```
class A{
    int m_a = 1;
    int m_b = 2;
    int m_c = 3;
    uint8_t m_d = 4;
public:
    A(int a, int b, int c, uint8_t d);
    A(const A& other);
    A();
    ~A();
    void Dump();
};
```

vocabluary types 浅尝

初步尝试 strong-typed enum, union 以及 variant

```
enum struct Tag : int16_t
{
    A_OBJ,
    INTEGER,
};
```

```
struct TagedPoorVariant
{
    Tag t;
    union
    {
        A m_A;
        int m_int;
    };
};

int main()
{
    TagedPoorVariant v1 = { Tag::A_OBJ, A{1,2,3,4}};
    v1.t = Tag::INTEGER;
    v1.m_int =  42;
    TagedPoorVariant v2 = { .t = Tag::INTEGER, .m_int = 42u};
    cout << "v1 has a size of " << sizeof(v1) << endl;
    cout << "v2 has a size of " << sizeof(v2) << endl;
    variant<int, std::string, A> v3;
    cout << "v3 has a size of " << sizeof(v3) << endl;
    v3 = "hello";
    cout << v3.index() << endl;
    v3 = 42;
    cout << v3.index() << endl;
    v3 = A{};
    cout << v3.index() << endl;
}
```

工业级 variant 实现
[tensorflow variant]

CDataBufer

请实现一个 DataBuffer 类

基本要求:

- 按照3法则实现 big four
- 写一些测试用例测试各种构造的行为是否正确, 特别是拷贝构造和拷贝赋值的语义是否正确
- 通过-fsanitize=address

```
class CDataBuffer
{
public:
    CDataBuffer(const std::string& name, A* data, unsigned int length);
    CDataBuffer(const CDataBuffer& db);
    CDataBuffer& operator = (const CDataBuffer& db);
    virtual ~CDataBuffer() = default;

    A* GetExtraData() { return m_pFoo; }
private:
    std::string m_dataName;
    unsigned int m_DataLength;
    unsigned int m_BufSize;
```

```
    A* m_pFoo;
};
```

进阶要求：

- 健壮性，将 trickster 函数放到可能出现异常的地方，比如申请新内存区域或者在工程实践中可能出现的地方，考察你的实现语义是否还正确
- 风格，尽量减少`-checks=cppcoreguidelines*`的警告
- 将 DataBuffer 内包含的指针改为指向一块连续裸数据的指针，并增加相应接口

```
int trickster()
{
    throw std::runtime_error("error");
}

class CDataBuffer
{
public:
    explicit CDataBuffer(const std::string& name, unsigned int bufSize =
DEFAULT_BUF_SIZE);
    CDataBuffer(const std::string& name, void* data, unsigned int length);
    CDataBuffer(const CDataBuffer& db);
    CDataBuffer& operator = (const CDataBuffer& db);

    virtual ~CDataBuffer() = default;

    bool Write(unsigned int pos, const void* data, unsigned int length);
    unsigned int Read(unsigned int pos, void* buf, unsigned int length) const;
    unsigned int GetBufferSize() const { return m_BufSize; }
    unsigned int GetDataLength() const { return m_DataLength; }

    FooTest& GetExtraData() { return m_foo; }
    const FooTest& GetExtraData() const { return m_foo; }
protected:
    bool CopyFrom(const CDataBuffer& db);
    void Release();
    bool AllocBuf(unsigned int size);
    bool CheckSpace(unsigned int length);

private:
    std::string m_dataName;
    unsigned char* m_dataBuf;
    unsigned int m_DataLength;
    unsigned int m_BufSize;

    FooTest m_foo;
};
```

工业级的 data buffer
[DataBufferHeap in lldb]
[image in chromium]