Part A – Give a short answer to the following questions. (40marks)

1. What is the type of the `std::cerr` object? (0.5 marks)

ostream

2. Write a #include statement to include the `std::shared_ptr` class in your program. (0.5 marks)

#include <memory>

3. What is the output of the following program? (2 marks)

```
#include <iostream>
#include <cstdlib>
#include <memory>

class base
{
public:
    base()
        {
            std::cerr << "base created" << std::endl;
        }

    virtual ~base()
        {
            std::cerr << "base destroyed" << std::endl;
        }
};

class derived : public base
{
public:
    derived()
        {
            std::cout << "derived created" << std::endl;
        }

    virtual ~derived()
        {
            std::cout << "derived destroyed" << std::endl;
        }
};

int main(void)
{
    std::unique_ptr<derived>d = std::make_unique<derived>();
    return EXIT_SUCCESS;
}
```

base created

derived created

derived destroyed

base destroyed

take half a mark off for each incorrect answer – they must be in this order.

4. Given that the following is valid c++ code:

```
std::unique_ptr<int> myint = std::make_unique<int>(5);
```

Does that mean that `int is a class?` If it is not a class, what is it? Be as precise as you can (1 mark)

no, it is not a class. It is a primitive or basic datatype. Half a mark for identifying it's not a class. Half a mark for identifying that it is a built in type.

5. What is the meaning of the noexcept keyword and how would you use it ? What happens if you throw an exception in a function marked noexcept? (2 marks)

labelling a function as noexcept guarantees that the function will never throw. This is a strong guarantee. If the function does throw, terminate() will be called and the function will never return.

6. Why doesn't the STL list class support random access? (1 mark)

while the STL list class supports sequential access of the elements in the list, it does not support random access because the elements are not contiguous.

1 mark for a similar answer to the one above.

7. Show an example of using the `std::stol` function to extract an integer from a string. Include any exception handling (2 marks)

```
long l;
try{
i = stol(str);
}
        catch(std::invalid_argument ia)
{
        std::cerr << "error: input was invalid." << std::endl;
}
```

1 mark for correct call to stol(). 1 mark for correct exception handling

8. Name the three classes introduced in c++11 that implement the safe pointer idea. Briefly explain the purpose of each type and explain its use (6 marks)

unique_ptr, weak_ptr, shared_ptr

unique_ptr defines a pointer to an object owned by a single class / function. Eg:

shared_ptr defines a pointer to a shared object such as when there are multiple pointers pointing to one object and you want too free the object once there are no pointers to it.

weak_ptr is a parent class for unique_ptr and shared_ptr and may point to either of these child classes.

9. Consider the following class declaration:

```
template<typename T>
class ptr_vector : private std::vector<std::unique_ptr<T>>
{
    std::vector<std::unique_ptr<T>> container;
    public:
    void add(std::unique_ptr<T>&& );
    virtual ~ptr_vector(void);
};
```

a) implement the function add() including any appropriate exception handling (4 marks).

b) implement the destructor for this class (2 marks).

You may assume that the add function and destructor and implemented inside the class declaration. (6 marks in total)

```
void add(std::unique_ptr<T> && item) throws ...
{
    container.push_back(std::move(item));
}
virtual ~ptr_vector(void){}
```

10. Consider the following code segment:

```
template<class T>
class tree
{
    class node
    {
        std::unique_ptr<node<T>> left;
        std::unique_ptr<node<T>> right;
        T data;
    };
};
```

We want to make left and right accessible to the tree without accessors and mutators. Outline two ways we may achieve this (6 marks)

first way: make node a struct. node is already private within tree so there is no harm in doing that.

Second way: make tree a friend of node. 3 marks each. This is a lot of marks for this but then I think it is an important concept.

11. It is common in C++ to inline a function. What are the two ways we can specify that a function should be inlined? (2 marks)

write the function inside the class declaration that it belongs to. Specify that a function should be inlined using the inline keyword.

1 mark for each.

13. In C, void pointers are used to implement generic programming. C++ provides similar functionality via templates. What benefits do they give us? Is there a runtime performance difference to using templates over void pointers? Which is faster and what cost do we pay for this speed? (5 marks)

Benefit: we can write a single class blueprint and multiple classes are generated at compile time from this (2 marks)

runtime performance: void pointers are slower as separate type information needs to be evaluated at runtime. (2 marks)

The cost for this is larger binary size due to multiple instances of a class template (1 mark)

14. Discuss the internal data representation that is typically used with a std::list, a std::vector and a std::set. Explain the relationship between the internal representation of each data structure and its expected performance, ranking each datastructure from slowest to fastest in terms of expected performance (6 marks)

A list is represented by having a series of list nodes with pointers going backwards and forwards. These nodes are located all over the place in memory. For this reason it is the slowest data structure. The std::set comes next in speed. This is implemented as a binary search tree with pointers to the left and right nodes. Once again, these are pointers to memory located elsewhere and are thus poisonous to cpu caching. A BST does however have the advantage of logarithmic search time which is much faster than the linear search time of the std::list. Finally we have the vector which has much shorter search and access times mainly due to our ability to cache data we are likely to lookup next, along with our ability to do random access.

15. Write a short program that reads in a line from a file called ints.txt that is delimited by commas and has integers stored between the commas. Tokenize this data and extract the numbers and store them and insert them into a vector. Tokenize this data and convert to integers to store in a vector. Now, save this data to a binary file. You should write out the number of integers in the vector followed by the data stored in the vector (20 marks) .

```
/* correct header file includes – 2 marks */
#include <iostream>
#include <fstream>
#include <boost/tokenizer.hpp>


int main(void)
{
    /* declare vector and files   - 3 marks */
    std::vector<int> ints;
    std::ifstream in("nums.txt");
    in.exceptions(std::ios::failbit | std::ios::badbit | std::ios::eofbit);
    std::string line;
```

```
    boost::char_separator<char> sep {","};
    try
    {
        in.clear();
        /* read data from file - 3 marks */
        while(!in.eof() && std::getline(in, line))
        {
            /* tokenize each line read in - 3 marks*/
            boost::tokenizer<boost::char_separator<char>>tok(line, sep);
            for( auto & s : tok )
            {
                /* convert to int - 3 marks */
                ints.push_back(stoi(s));
            }
        }
        in.close();
        /* open binary file - 2 marks */
        std::ofstream out("nums.bin", std::ios::binary);
        int size = ints.size();
        /* write size followed by vector contents - 3 marks */
        out.write((const char *)&size, sizeof(int));
        out.write((const char *)&ints[0], sizeof(int) * size);
        out.close();
    }
    /* correct exception handling - 3 marks */
    catch(std::exception & ex)
    {
        if(!in.eof())
        {
            std::cerr << ex.what() << std::endl;
        }
    }
}
```

Part B: Operator overloading (60 Marks)

We can model a game board as a vector of vector of cells (a 2d vector of board locations) where each cell may contain a red piece, a white piece or be empty.

1. Write the declaration (not the implementation) of the class board. Assume that this class is defined in a file called board.h. Include the prototypes for the following methods (6 marks):

a) A default constructor with two optional integer arguments set to the default value of 8. This should create a board where the first argument is the width and the second argument is the height.

b) A copy constructor.

c) A move constructor for a board

d) the assignment operator for this class.

e) A conversion operator to the int type so that whenever an object of type board fraction is assigned to an integer, it returns the number of squares on the board. For example an 8x8 board would return 64.

f) The prototype for a static member function for finding whether a board with the width and height specified would be square (same number of squares wide as high.

g) The prototype for overloading operator<<(). Place this in the correct location and insert any other statements required to make this work. Please note that if other types need to be output you should write operator functions in preference to handling their output within this functions.

h) A less than operator for comparing boards which returns true when the current board has less area than the one it is being compared to.

i) The prototype for a destructor.

```
#include <vector>
#include <iostream>
enum class cell_type
{
    WHITE, RED, EMPTY
};

struct cell
{
    cell_type contents;
};

class board
{
    int width, height;
    std::vector<std::vector<cell>> cells;
    public:
    board(int=8,int=8);
```

```
        board(const board&);
        board(const board&&);
        board operator=(const board&);
        operator <(const board&);
        operator int();
        static int is_square(int, int);
        friend std::ostream& operator<<(std::ostream&, const board&);
        virtual ~board(void);
    };
```

deduct a mark for each missing component and ½ mark for each incorrect one.

Note: Assume all functions implemented below are implemented in a file called board.cpp that includes the above mentioned file board.h

2. Implement the default constructor for the fraction class as specified in 1 a). (4 marks)

```
board::board(int width,int height) : m_height(height),
    m_width(width), cells(std::vector<std::vector<cell>>(m_height))
{
    int count;
    for(count = 0; count < m_height; ++count)
    {
        cells[count] = std::vector<cell>(m_width);
    }
}
```

3. Implement the copy constructor specified in 1 b).(6 marks)

```
board::board(const board& oboard) : m_height(oboard.cells.size()),
    m_width(oboard.cells[0].size())
{
    int ycount, xcount;
    for(ycount = 0; ycount < m_height; ++ycount)
    {
        cells[ycount] = std::vector<cell>(m_width);
        for(xcount = 0; xcount < m_width; ++xcount)
        {
            cells[ycount][xcount] = oboard.cells[ycount][xcount];
        }
    }
}
```

4. Implement the move constructor specified in 1 c) (4 marks)

```
board::board(const board&& oboard) : m_height(std::move(oboard.m_height)),
    m_width(std::move(oboard.m_width)), cells(std::move(oboard.cells))
{
}
```

5. Implement the assignment operator specified in 1 d) (8 marks)

```
board board::operator=(const board& oboard)
{
    int ycount, xcount;
    board newboard(oboard.m_width, oboard.m_height);
    newboard.cells = std::vector<std::vector<cell>>(oboard.m_height);
    newboard.m_height = oboard.m_height;
    newboard.m_width = oboard.m_width;
    for(ycount = 0; ycount < oboard.m_height; ++ycount)
    {
        for(xcount = 0; xcount < oboard.m_width; ++xcount)
        {
            newboard.cells[ycount][xcount] = oboard.cells[ycount][xcount];
        }
    }
    return *this;
}
```

6. Implement the conversion operator specified in 1 e).(5 marks)

```
board::operator int()
{
    return m_width * m_height;
}
```

7. Implement the static member function specified in 1 f).(4 marks)

```
bool board::is_square(int x, int y)
{
    return x == y;
}
```

8. Implement the operator<<() function specified in 1 h).(10 marks)

```
std::ostream& operator<<(std::ostream& out, const cell& mycell)
{
    std::string content_strings[] =
    {
        "WHITE", "RED", "EMPTY"
    };
    out << content_strings[int(mycell.contents)] ;
    return out;
}

std::ostream& operator<<(std::ostream& out, const board& oboard)
{
    unsigned x, y;
    auto & cells = oboard.cells;
    for(y = 0; y < cells.size(); ++y)
    {
        for(x = 0; x < cells[0].size(); ++x)
```

```
            {
                std::cout << cells[y][x] << ", ";
            }
        }
        return out;
}
```

9. Implement the less than operator specified in 1h). (3 marks)

```
bool board::operator<(const board& other)
{
    int area = m_width * m_height;
    int oarea = other.m_width * other.m_height;
    return area < oarea;
}
```

10. Implement the destructor specified in 1 i). (4 marks)

```
fraction::~fraction() throw()
{
}
```

11. Write a lambda function that accepts two const cell references as arguments and returns true when the first cell is "less than" the second cell. The ordering of "less than" for a cell shall be(from lowest to highest): EMPTY, WHITE, RED. This lambda should not capture anything from its environment (6 Marks).

```
auto mylambda = [](const cell & a, const cell & b)
    {
        switch(a.contents)
        {
            case cell_type::EMPTY:
            {
                if(b.contents == cell_type::WHITE ||
                    b.contents == cell_type::RED)
                {
                    return true;
                }
                return false;
            }
            break;
            case cell_type::WHITE:
            {
                if(b.contents == cell_type::EMPTY)
                {
                    return false;
                }
                return true;
            }
            break;
            case cell_type::RED:
            {
                return false;
            }
        }
        return false;
```

```
};
```