Part A – Give a short answer to the following questions. (60 marks)

1. What is the type of the `std::cin` object? (1 mark)

   ostream

2. Write a #include statement to include the `std::unique_ptr` class in your program. (1 mark)

   #include <memory>

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

   ```
   std::string s = "fred";
   ```

   Does that mean that "`fred`" is a std::string? If it is not a string, what is it? Be as precise as you can (1 mark)

   no, it is not a string. It is a char array. Half a mark for identifying it's not a string. Half a mark for identifying it is a char array.

4. What is the default visibility of c++ class members? (1 marks)

   private

5. If we have a common parent class A, and two child classes, B and C, how can I safely downcast from A to B or C in order to access methods only defined in one of the child classes? (1 mark)

   use dynamic_cast

6. Given that the STL `map` supports the [] operator for accessing elements, does this mean that it has vector / array like performance for accessing its elements? Why is/is not that the case? (3 marks)

   the map class does not have vector / array like performance. It is typically implemented as a red black tree which while it has reasonable performance it is not as good as a vector or an array.

   1.5 mark for no. 1.5 mark for the reasoning.

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

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

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

8. Name the three classes introduced in c++11 that implement the safe pointer idea. (3 marks)

   unique_ptr, weak_ptr, shared_ptr

9. The following code generates a compiler error. What is the reason that this is a compiler error (not just the location but the actual reason that it is a compiler error)? You may assume that there is a validly defined widget class in scope.(1 mark)

   std::vector<std::vector<widget&> > widgets;

   The source of the error is the use of a widget reference rather than a widget or widget*. A reference must refer to an existing object.

   ½ mark for pointing out the location of the error and ½ mark for pointing out the reason that it is an error.

10. What problem exists in the following code? How could we avoid it?(3 marks)

```
class foo
{
	int i;
	public:
		foo() : i(0) {}
};
class bar : public foo
{
	friend ostream& operator<<(ostream&, const bar&);
};
ostream& operator<<(ostream& out, const bar& b)
{
	out << b.i;
	return out;
}
```

   while operator<<() is a friend of bar, it is not a friend of foo. As such, given that i is private, it is not accessible. We could avoid this problem by making i protected or making operator<<() a friend of foo. 1 mark for realising that this is an issue with the visibility of i. 1 mark for discussing friendship correctly and 1 mark for suggesting a reasonable solution.

11. What does it mean to "inline" a function? How is this achieved and what benefit do we get from doing so. Is there a cost or down-side to doing this? (5 marks)

    if we define a function inside a class declaration, the function is inlined which means that a call to the function is replaced by the body of the function which is much faster than calling the function due to the overheads of copying data onto the stack and jumping elsewhere in memory. The cost of doing this however is that the executable may be larger due to the function's body being replaced inline.

    3 marks for definition of inlining, 1 mark for a benefit and one for a cost.

12. What object-oriented concept is enabled through the use of the "virtual" keyword? Explain What benefit(s) does this give us? Are there any costs? If so, outline one of these. (4 marks)

    The virtual keyword enables polymorphism in classes and structs.

    the facility by which different objects in the same inheritance hierarchy display different behaviour for the same function call.

    The benefit of doing this is that the actual function called is bound at runtime rather than at compile time.

The cost of doing this is that there are additional lookups to the virtual function table at runtime which results in slower execution.

1 mark for definition

1 mark for mentioning the virtual keyword

1 a mark for benefit, 1 mark for cost.

13. In the C programming language, generic programming would be achieved using void pointers. C++ has introduced a better way to achieve this. What is that better way? Why is it better? Is there a runtime cost to this better way? Why/ why not? (6 marks)

C++ has introduced the concept of templates as a simpler way to do generic programming.

A template is a blueprint for a class or a function. It allows us to do generic programming with some degree of type safety. There is no runtime cost to their use however they result in larger executables due to multiple copies of the template being generated – one for each type that the template is used with.

1 mark for mentioning templates

1 mark for mentioning that a template is a blueprint / template for a class/function.

1 mark for mentioning type safety

1 mark for mentioning the generation of larger executables.

1 mark for mentioning that there is no runtime cost to their use.

14. 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.

15. There is a memory access problem in this code segment below. Identify the problem and a solution to the problem. Only rewrite those lines of code you need to to fix the problem. (6 marks)

```
#include <cstdlib>
#include <exception>
#include <iostream>
#include <sstream>
#include <vector>

class outofrange : public std::exception
{
  int offender_;
public:
  outofrange(int offender) : offender_(offender) {}
  const char * what() const throw()
  {
    std::ostringstream oss;
    oss << offender_ << " is out of range";
    return oss.str().c_str();
  }
};

int main(void)
{
  std::vector<int>ints(40);
  /* some code to initialise the elements of the vector here */
  int input;
  std::cout << "Please enter an index to get the value of: " << std::endl;
  std::cin >> input;
  try
    {
      if(input > int(ints.size()))
       {
         throw outofrange(input);
       }
    }
  catch(outofrange& oor)
    {
      std::cerr << oor.what() << std::endl;
    }
  return EXIT_SUCCESS;
}
```

The problem is in the function outofrange::what()
The function is returning a pointer that is part of the ostringstream's string object.
This pointer is no longer valid once we return from this function.
Two possible solutions are to declare the ostringstream object as class level variable and then it would be cleaned up by the destructor or we could do the following:

```
  const char * what() const noexcept
  {
    std::ostringstream oss;
```

```
    oss << offender_ << " is out of range";
    std::string result = oss.str();
    char * retval = new char[oss.str().length() + 1];
    std::strcpy(retval, oss.str().c_str());
    return retval;
  }
```

3 marks for a full identification of the problem and 3 marks for a full identification of the solution.

16. Name three types of iterators are supported by the Standard Template Library. What types of iteration do each type of iterator allow? (6 marks)

all iterators can be incremented

input iterator: supports equality comparisons and can be dereferenced as an rvalue (data can be assigned but cannot be assigned to). (iteration over an input stream)

Output iterator: can be dereference as an lvalue but not an rvalue. (iteration over an output stream)

forwards iterator: iterate forwards over a collection.

Reverse iterator: iterate backwards over a collection.

Bidirectional – can iterate forwards or backwards across a collection

Random access iterators: forwards, backwards and add to by some int other than 1. eg: it+=3;

17. Unlike more modern object-oriented languages such as java, c++ arrays of objects are stored contiguously in memory. What advantages does this provide in terms of performance. Name a modern hardware component that assistts in this process and explain in a sentence or two how it helps. (8 marks)

As objects are stored next to each other in arrays of objects and this helps with performance as when we we retrieve one object we can also retrieve the objects stored next to them. Likewise, it is more likely when we need access to an object that it is still in cache and thus we avoid requests to physical memory which are orders of magnitude slower.

The hardware components that they may mention are either the prefetcher or cpu cache. When a request is made to retrieve an object from memory, the prefetcher will retrieve the objects next to the requested object and store them in cache. When the cpu requests an object, it will firstly check in the cache to see if it is there and if it is, it avoids the expensive round trip of retrieving objects from memory.

4 marks for the discussion of the importance of contiguity.

4 marks for the discussion around either cache or the prefetcher.

18. There is a linker error and a compiler error in the following program. Identify the errors and how we might fix them. (6 marks)

in foo.h:

```
#include <iostream>

#include <cstdlib>

int i;

class foo

{

        int t;

        public:

        foo(int & _t) : t(_t) {}
```

```
        const int & get_t() const { return t; }
};
```

in foo.cpp:

```
#include “foo.h”
int i;
int main(void)
{
        foo f(3);
        std::cout << f.get_t() << std::endl;
        std::cout << i << std::endl;
        return EXIT_SUCCESS;
}
```

errors: i is defined twice – that means that there is two lots of memory allocation for i. They need to get the point about memory allocation and multiple definitions to get full marks here.

foo's constructor specifies a reference but 3 is an r value and as such we cannot take the address of it and therefore we cannot have a reference to it.

Each of these points is to be marked out of 3: 1 mark for getting the right location of the error, 1 mark for a correct understanding of why it is an error and 1 mark for the correct fix.

Part B: Anatomy of a Class (60 Marks)

We wish to model bicycle parts for a company selling these. As part of this process we will use various advance features of c++ classes to keep track of information on each part. A bicycle part has a part number, a weight, a price, a brand name and a country of origin.

1. Write the declaration (not the implementation) of the class bicycle. Assume that this class is defined in a file called bicycle.h. If you feel you need to create other classes or structs to complete the functionality here, you should feel free to go ahead and do so. Include the prototypes for the following methods. Note that not all are required for this class – you are required to make these to show the concepts you have learnt in this course (8 marks):

   a) A default constructor.

   b) A constructor that initialises all elements specified in the above description.

   c) A copy constructor.

   d) the assignment operator for this class.

   e) A conversion operator to the 'double' type

   f) The prototype for a static member function called total_cost which takes in a vector of bicycle parts and returns the total cost for these parts.

   g) Make the operator<<() a friend of this class.

   h) The prototype for a destructor.

```
struct price
{
        bool sign;
        int dollars;
        int cents;
};
class bicycle_part
{
        std::string partno;
        double weight;
        price myprice;
        std::string brandname;
        std::string origin;
        public:
                /* constness is optional */
                bicycle_part();
                bicycle_part(const std::string&, double, const price&, const std::string&,
                std::string&);
                bicycle_part(const bicycle_part&);
                bicycle_part operator=(const bicycle_part&);
                operator double();
                static double total_cost(const std::vector<bicycle_part>&);
                virtual ~bicycle_part() noexcept;
                friend ostream& operator<<(ostream&, const bicycle_part&);
};
```

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

In the following questions, you will be implementing the components of the class that were specified above. Each of these members should be assumed to be implemented in the .cpp file for this class – called bicycle_part.cpp

2. Implement the default constructor specified in 1 a) to initialize all values to sensible defaults. (6 marks)

```
bicycle_part::bicycle_part(void)
    : partno(“”), _weight(0), price({0,0}), brandname(“”), origin(“”)
{
}
```

deduct a mark for each substantial aspect that is incorrect

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

```
bicycle_part::bicycle_part(const std::string& _partno, double _weight, const price&
_price, const std::string& _brand, std::string& _origin)
    : partno(_partno), weight(_weight), myprice(_price), brandname(_brand),
    origin(_origin)
{
}
```

deduct a mark for each substantial aspect that is incorrect

4. Implement the copy constructor specified in 1 c).(6 marks)

```
bicycle_part::bicycle_part(const bicycle_part& part) : partno(part.partno),
weight(part.weight), price(part.price), brandname(part.brandname), origin(part.origin)
{
}
```

again, deduct a mark for each incorrect element.

5. Implement the assignment operator specified in 1 d) (6 marks)

```
bicycle_part & bicycle_part::operator=(const bicycle_part& part)
{
  bicycle_part copy;
  partno = part.partno;
  weight=part.weight;
  price=part.price;
  brandname = part.brandname;
  origin = part.origin
  return *this;
  //look for the student returning a reference to a local value as that won't work (-2 for
  that) otherwise subtract a mark for each thing they do incorrectly.
}
```

6. Implement the conversion operator specified in 1 e). This conversion operator should simply return the weight of the bike part. (8 marks)

```
bicycle_part::operator double()
{
  return weight;
}
```

7. Implement the static member function specified in 1 f).(10 marks)

```
double bicycle_part::total_cost(const vector<bicycle_part>&  parts)
{
        price total = {0,0};
}
```

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

```
ostream& operator<<(ostream& out, const fraction& part)
{
   out << part.partno << “,” << part.weight << “,$” << part.price.dollars << “.”
       << part.price.cents << “,” << part.brandname << “,” << part.origin ;
   return out;
}
```

deduct 2 marks for each thing incorrect.

9. Implement the destructor specified in 1 i). (2 marks)

if the student puts anything in this function it tends to suggest they don't know the purpose of a destructor and so they don't get the marks.

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