Marking Scheme

1. (a) (i)

10 marks

```
void Modify_list(list<int> &l) {
     for (auto &x : l)
          ++x;
}
```

Marking: signature [2, including 1 for ref], the body [3]

An explicit iterator is equally acceptable:

```
void Modify_list(list<int> &l) {
     for (list<int>::iterator p = l.begin(); p != l.end(); ++p)
          ++(*p);
  }
```

```
(ii)

int Smallest(const list<int> &l){
     auto it=l.cbegin();
     int min=*it;
     for(auto x:l)
          if(x<min)
               min=x;
     return min;
}
```

Marking: signature [2, including 1 for const ref], the body [3]

(b) (i)

10 marks

```
bool odd(int x) { return x % 2 == 1;}

void update_odd(list<int> &l) {

     auto p = find_if(l.begin(), l.end(), odd);

     if (p != l.end())

          *p = 1;

}
```

Marking: odd [2, 1 for signature, 1 for body], update_odd[signature 3, including 1 for ref, calling find_if 3, including 2 for l.begin and l.end, if they have used l.cbegin or l.cend deduct 2, if statement and its body 2].

An explicit iterator is equally acceptable (instead of auto)

(ii) 10 marks

```
bool Is_even(int x){
  return x%2 == 0;
}

int Count_list(const list<int> &l){
     auto p = find_if(l.cbegin(),l.cend(),Is_even);
     if ( p == l.cend() )
         return 0;
     return count_if(p,l.cend,odd);
}
```

Marking: signature [3, including 2 for const ref], find_if [3], count_if [3] and [1] for Is_even. Odd was written in previous part.

(c) (i) Expect something like: 10 marks

```
class music:public multimedia{
  public:
       music(const string &s): multimedia(s){  }
       virtual void play() const{
            cout<<"This is music: "<<description();
       }
  } ;
```

Marking: class and inheritance [2], public: [1] constructor [3], virtual function signature [2], its body [2],

(ii) 2 marks for an invalid answer like 10 marks

```
vector<multimedia> items;
```

because we cannot create `mutimedia` objects.
Full marks for an answer using shared pointers:

```
vector<shared_ptr<multimedia>> items;
items.push_back(make_shared<music>(" Louie Louie"));
items.push_back(make_shared<film>(" Casablanca"));
items[0]->play();
items[1]->play();
```

An answer involving pointers

```
vector<multimedia *> items;
```

gets 6 marks, plus 2 more if they say they will use `delete` on each element of the container when finished.

2. (a) i and ii

20 marks

```
class Vehicle{
       string _name;
       int _value;
       string _color;
 public:

     Vehicle(const string &n,int v, const string
     &c):_name(n),_value(v),_color(c){}
     const string &name() const {return _name;}
     int value() const {return _value;}
     const string &color() const {return _color;}
     };
```

Marking i: members [2], `public` [1], constructor [7] (1for const and ref)

Marking ii: accessors [10], 3 for each and 1 for const and ref. Additional const and ref for the `int` are optional.

(b)

30 marks

```
class parking{
   list<Vehicle> l;
public:
   void add(const Vehicle &v){
             l.push_back(v);
        }
   list<Vehicle>::size_type number()const{
             return l.size();
        }
   int total_value() const{
             int sum=0;
             for(const auto &x:l)
                  sum +=x.value();
             return sum;
        }
   int number_given_color(const string &s) const{
        auto equal = [&s] (const Vehicle &x) { return x.color()
== s; };
                return count_if(l.cbegin(), l.cend(), equal);
                }
          };
```

Marking: [2] for defining the class and a collection (list, vector..)

Marking i: 2 for the signature (1 for the const ref) and 2 for the body

Marking ii: 2 for the signature (1 for the return type) and 2 for the body

Marking iii: 3 for the signature 7 for the body, they can use iterators for the body

Marking iv: 3 for the signature (1 for const ref) and 4 for the equal and 3 for the count_if, they can use directly the lambda expression in the count_if