MA1262 - original exam and solutions, final exam may be slightly different.

1. (a) (4) How would this programme have to be changed

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

using namespace std;

int main()
{
      cout<<"Of man's first disobedience"<<endl;
}
```

if the line `using namespace std;` was removed.

Solution: If the `using namespace std;` was removed you wouldn't be in the `std;` namespace so the namespace for the `cout` and `endl` would have to be indicated explicitely,

```
std::cout<<"Of man's first disobedience"<<std::endl;
```

(b) (3) What will this programme output

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

using namespace std;

int main()
{
      int a=0;

      cout<<a<<endl;
      cout<<a++<<endl;
      cout<<++a<<endl;
}
```

Solution: The output will be 0 0 2 with carriage returns inbetween since the first increment is a postincrement, it only happens after the value is return, the second increment is a preincrement, it happens before the value is returned.

(c) (3) What is the difference between `endl`, `flush` and \ `n`?

Solution: `flush` flushes the stream, so everything queued for outputing is outputted, \ `n` prints out a carriage return, so subsequent output appears on the next line, `endl` does both.

(d) (6) What will this programme output

```
#include<cstdlib>
#include<iostream>
#include<cmath>

using namespace std;

int main()
{
      int a=3;
      int b=5;
      double c=b/a;

      cout<<c<<endl;

      c=double(b)/a;

      cout<<c<<endl;
      cout<<int(c)<<endl;
      cout<<floor(c)<<endl;
      cout<<ceil(c)<<endl;

      c=-double(b)/a;

      cout<<c<<endl;
      cout<<int(c)<<endl;
      cout<<floor(c)<<endl;
      cout<<ceil(c)<<endl;

}
```

Solution: 1 1.6667 1 1 2 -1.66667 -1 -2 -1 with returns inbetween. `b/a` cast as an `int` since it is the division of two `int`s, when it is cast as a `double` it is already one. However, dividing a `double` by an `int` gives a `double`. Casting to an `int` cuts off the decimal, doing `floor` gives the integer less than it and `ceil` gives the one above it.

2. (a) (6) What is wrong with this programme

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

using namespace std;

int main()
{
      double a=0;
      if(a=0)
            cout<<"I want this printed out"<<endl;
}
```

What will the output be and why? How would you correct this programme?

Solution: The problem is the `=` instead of `==` in the `if` statement. Some compilers won't compile this, but if it does, the assignation returns the value assigned, in this case `0` which casts to `false` so nothing is printed. Obviously the way to change it is to change the assignation `=` to the boolean `==`.

(b) (6) What will the output of this program be

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

using namespace std;

int main()
{
      double a=0;
      if(a==1&&cout<<"I am in your base ")
```



```
            cout<<"shootin your d00dz"<<endl;
}
```

What will the output be? What would it be if the line `double a=0` was changed to `double a=1`? Why?

Solution: So `a` is not one, so the boolean in the `if` statement fails on the `a==1` and, because and is lazy, the second line is not evaluated, however, if the `a` is changed to one, the first part is true, so the second is evaluated, evaluating a `cout` causes it to print out and evalate as `true` if it is successful, so the program will print `I am in your base shootin your d00dz`.

(c) (4) If `a` and `b` are two `bool`s give the value of the following for the four possible combinations of `a` and `b` being `true` and `false`: `(a&&b)`, `(a||b)`, `(a&&!b)`, `(a&&(b||!b))`.

Solution:

For TT we have `(a&&b)` T, `(a||b)` T, `(a&&!b)` F and `(a&&(b||!b))` T. For TF we have `(a&&b)` F, `(a||b)` T, `(a&&!b)` T and `(a&&(b||!b))` T. For FT we have `(a&&b)` F, `(a||b)` T, `(a&&!b)` F and `(a&&(b||!b))` F. For FF we have `(a&&b)` F, `(a||b)` F, `(a&&!b)` F and `(a&&(b||!b))` F.

(d) (4) What is the output of this programme

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

using namespace std;

int main()
{
      double a=1;
      if(a==0)
            cout<<"red lorry"<<endl;
            cout<<"yellow lorry"<<endl;
      cout<<"green lorry"<<endl;
}
```

Solution: The trick here is that the second `cout` is not in the `if` block since there are no curly brackets, so the output will be

```
yellow lorry
green lorry
```

3. (a) (6) This programme won't compile, why not? Correct it so that it compiles.

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

using namespace std;

int main()
{
      double a=6.5;
      print(a);
}

void print(int b)
{
     cout<<b<<endl;
}
```

What will the output be and why?

Solution: So there is no prototype for the function, the line `void print(int b)` needs to be added before the `int main()`. If that's done the output will be `6` since the call will cast the value to an `int`.

(b) (7) What will the output of this programme be.

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

using namespace std;

int prints_stuff(int a,int & b);

int main()
{
```



```
      int a=4;
      int b=-2;
      cout<<prints_stuff(a,b)<<" "<<a<<" "<<b<<endl;
}


int prints_stuff(int b,int &a)
{
     cout<<b<<endl;
     a++;
     b++;
     return a+b;
}
```

What will the output be and why?

Solution: So this is all about confusing you with things changing names, what matters is the place things are in, so `a` is found and `b` is `-2`, the `cout prints the output of the function, inside the function b is the name of the first argument of the function whereas in main this is a; when the function is called it prints out its b giving`

`4`

`one is added to both variable and their sum is returned, so the cout prints out a four again, ie (4+1)+(-2+1), one has been added to the second argument of the function in the main since that variable is passed by reference, the second argument is the b in main so`

`4 4 -1`

(c) (5) Give a quick description of each line of output.

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

using namespace std;
```

```
int main()
{
      bool a_true_thing=true;
      vector<bool> a;

      cout<<a.size()<<endl;

      a.push_back(a_true_thing);
      a.push_back(!a_true_thing);
      a.push_back(a_true_thing);
      a.push_back(a_true_thing);

      cout<<a.size()<<endl;

      for(unsigned int i=0;i<a.size();i++)
            {
                  cout<<a.at(i)<<endl;
                  a.at(i)=!a.at(i);
            }

      cout<<"\n";

      for(unsigned int i=0;i<a.size();i++)
            cout<<a[i]<<endl;

}
```

Solution: So the vector a is TFTT, the F coming from the !. When it prints out the size it give four, then it prints out each entry giving 1 0 1 1 with returns corresponding to the TFTT, it nots each of them as it does so, so after the extra line from the \ n it prints out 0 1 0 0 with returns.

(d) (2) What is the difference between a.at(i) and a[i] in the above programme.

Solution: a.at(i) does a range check, a[i] does not, so with the

former calling the vector for an index not in its range will give an error, with the latter it may give an error, but a more obscure one, or it might just mess up the result of the program. The range check does make the former slower.

4. (a) (5) What is meant by a *class*, a *class method* and an *instance of a class*?

   Solution: A class is a user defined datatype, a method is a function built into that datatype, an instance is a variable declared for the class.

   (b) (5) Describe the output of this code

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

using namespace std;

class Battery
{
      public:
            Battery(double charge){this->charge=charge;}
            Battery(){charge=1;}
            void deplete(){charge=0;}
            void recharge(){charge=1;}
            double get(){return charge;}
            void print(){cout<<charge<<endl;}
      private:
            double charge;
}

int main()
{
       Battery aa(0.5);
       Battery  b();
       Battery  c;

       aa.print();
       b.print();
```

```
        c.print();

        aa.recharge();
        b.deplete();

        aa.print();
        b.print();

}
```

Solution: So `b` and `c` are both declared with the default constructor, so they both have `charge` of one, `aa` is declared with the other constructor and has a `charge` of `0.5`. Hence the first block of `.print()`s prints out 0.5 1 1 with returns, next `aa` is charged and `b` depleted, so the output is 1 0.

(c) (5) Find and correct four mistakes in this code:

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

using namespace std;

class Bungee
{
      public:
            Bungee(int a);
            int set_a(int a);
            int get_a();
      private:
            int a;
}

Bungee::Bungee(int a){this->a=a;}
set_a(int a){this->a=a;}
get_a(){return a;}

int main()
{
       int b=2;
```



```
       Bungee stretchy_cord;

       stretchy_cord.set_a(b);

       cout<<stretcjy_cord.get_a();

}
```

Solution: So

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

using namespace std;

class Bungee
{
      public:
            Bungee(int a);
            int set_a(int a);
            int get_a();
      private:
            int a;
}; //";" added

Bungee::Bungee(int a){this->a=a;}
void Bungee::set_a(int a){this->a=a;}  // "void" and "Bungee::" adde
int Bungee::get_a(){return a;}  // "int" and "Bungee::" added

int main()
{
       int b=2;
       Bungee stretchy_cord; //there is no default constructor, so e

       stretchy_cord.set_a(b);

       cout<<stretcjy_cord.get_a(); //should be stretchy, this was a

}
```

(d) (5) What is a *template*, what is the output of the following programme and what are all the angle brackets for?

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

using namespace std;

template <class T>
T GetMax (T a, T b)
{
      T result=b;
      if (a>b)
           result=a;
      return result;
}

int main ()
{
      int i=5, j=6, k;
      k=GetMax<int>(i,j);

      double x=6.5; y=2.6, z;

      z=GetMax<double>(x,y);
      cout << k << endl;
      cout << z << endl;

}
```

Solution: A template stands in for a datatype, the angle brackets give the data type argument.