FE 545 - Assignment 4

Question 1)

```
#include <iostream>
#include "time.h"

int main(int argc, const char * argv[]) {
    // insert code here...
    clock_t start,finish;
    start=clock();
    int n=10000;
    size_t mySize = 1000;
    for(int i=1;i<=n;i++){
            double* data = new double[mySize];
    }
    finish=clock();
    std::cout<<"The number of times loop ran for " << n <<"\n";
    std::cout<<"The size of array is " << mySize <<"\n";
    std::cout << ((double)(finish-start))/CLOCKS_PER_SEC<<"\n";
    return 0;
}
```

Using new slows up the program, every time a new is declared the compiler allocates new space in memory. So with every new compiler has to look for new space in memory and mark it as being used, this helps when we are deleting the variable but the amount of work involved every time a new variables in created is a lot and thus more time is consumed as seen in the above code and results.

```
The number of times loop ran for 10000
The size of array is 20
0.000718
Program ended with exit code: 0
```

```
The number of times loop ran for 10000
The size of array is 100
0.000792
Program ended with exit code: 0
```

```
The number of times loop ran for 10000
The size of array is 1000
0.000907
Program ended with exit code: 0
```

Question 2)

Auto_ptr

The auto_ptr class template describes an object that stores a pointer to a single allocated object that ensure that the object to which it point gets destroyed automatically when control leaves scope.
So the work of Auto_ptr is to free the data to which it points whenever the pointer itself gets destroyed. Because the pointer is available locally it will be destroyed automatically when the function is exited regardless of whether the exit is normal or caused by exceptions and thus prevent any kind of memory leaks.

Copying of Auto_ptr

```
std::auto_ptr<int> a;

std::auto_ptr<int> b;

a = std::auto_ptr<int> (new int);

*a = 11;

b= a;
```

In the above case we have 2 auto_ptr, the object on the left hand side(**b**) is now incharge of the pointer and will free the memory when when **b** is destroyed. **a** in the other hand have null pointer after transfer of pointer.

```
#include <iostream>

using namespace std;

class Man
{
public:
    Man(int v = 0) : Value(v) { cout << "Constructor: " << Value << endl; }
    ~Man() { cout << "Destructor: " << Value << endl; }
    void setMan(double v) { Value = v; }
private:
    int Value;
};

int main()
{
    auto_ptr<Man> ptr(new Man(10));
    (*ptr).setMan(15);
    return 0;
}
```

```
Constructor: 10
Destructor: 15
Program ended with exit code: 0
```

Ptr is a local automatic variable in **main()**, **ptr** is destroyed when main ends. The auto_ptr destructor forces a delete of the Man object pointed to by **ptr**, which in turn calls the Man class destructor. The memory that Man occupies is released.

```
#include "AutoPtrTest.hpp"
#include <iostream>
#include <memory>

using namespace std;

class Man
{
private:
    double value;
    auto_ptr<Man> ptr;

public:
    Man(double len) { value=len;cout << "constructor: " << value << endl; }
    Man(Man& obj):ptr(_obj.value.get()?(new Man(*_obj.value.get())):0){
        // ptr=*obj;
        // ptr(obj.Value);
        // ptr=obj.Value;
    }

    ~Man() { cout << "destructor: " << value << endl; }
    //void setMan(double d) { Value = d; }

};

int main()
{
    //auto_ptr<Man> ptr(new Double(3.14));
    Man value(10);
    //(*ptr).setMan(15);
    return 0;
}
```

Errors

```
ptr=*obj;        Indirection requires pointer operand ('Man' invalid)
ptr= obj;        No viable overloaded '='
ptr(obj.value);  Type 'auto_ptr<Man>' does not provide a call operator
ptr(*obj.value); Indirection requires pointer operand ('double' invalid)
```

After repeated trying I was not able to assign values to auto_ptr class member value. I searched online and these were the alternate I found

1) Making use of unique_ptr, I was earlier trying to pull it off with C++11 which doesn't support auto_ptr any more than I shifted to C++98 still wasn't able to find any solutions

2) Making use of two different classes auto_ptr and auto_ptr_ref, I found a solution which gave a tricky solution as mentioned in the post on Stack Overflow, I have given the link below

https://stackoverflow.com/questions/4514124/how-could-one-implement-stdauto-ptrs-copy-constructor

3) Why was auto_ptr removed from the further updates ?

https://stackoverflow.com/questions/2404115/is-auto-ptr-deprecated

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1856.html#20.4.5%20-%20Class%20template%20auto_ptr