1 CHARS AND VECTORS

1.1 The first char is initialised using a string literal and is terminated with the '\0' character, whilst the yourName is initialised as an array of chars and is not terminated with a '\0'.

1.2 Vector <int>. number (4);

1.2.1 We are declaring a vector of type int with a size of 4;

1.2.2 Number.push_back(11);

1.2.3 11

2 RECURSION

2.1 .olleH

2.2 It reverses text that the user enters provided the user enters a full stop.

2.3 We will have an unconditional recursion and the recursion will never exit until we run out of memory.

3 POINTERS

3.1 A dangling pointer that points to invalid data or a pointer to storage which is no longer allocated. Be very careful when using the delete operator. They can be disastrous because they are usually not picked up before compilation, they are difficult to find and rectify and usually, results in strange anomalies which are difficult to find at run time. Remember, run time errors are the worst kind of errors.

3.2 Which is problematic?

1. Int *p1, p2 (p2 has not been declared as a pointer but is used as such in line 4)
4. p2 = p1 (you can't assign a pointer to a normal int variable, you can only assign the address of an int variable by using the address of & operator or add assigning a value to the variable that int is pointed to by using the *dereferencing operator)

3.3 Int *p1, *p2;

3.4 Int array_size = 10; int *a; a = new int [array_size]; cout << &a;

3.4.1 We use the new operator to dynamically allocated memory to be used at run time. The new operator returns the starting address of the allocated space. It is difficult to make use of it like this because the allocated space doesn't have a name, that is why we store the address in a pointer.

3.4.2 delete a;

3.4.3 it will display the address of a.

4 SHARES CLASS WITH ARRAY

4.1 Share.h

```
#ifndef SHARES_H
#define SHARES_H
#include <iostream>
#include "share.h"
#include <string>
#include <fstream>
#include <cstdlib>

using namespace std;


class Shares
{
   public:
      Shares();
      Shares(string pCompany, int pNrShares, float pShareValue, double pPastUnitValues[5]);
      ~Shares();
      string getCompany();
      int getnrShares();
      float getunitValue();
      friend bool operator==(const Shares &s1, const Shares &s2);
      void updateShares(const Shares & s1);
      friend istream & operator >> (istream& ins, Shares & s);
      friend ostream& operator << (ostream& outs, Shares & s);


   protected:

   private:
      string company;
      int nrShares;
      float unitValue;
      double pastUnitValues[5];
};

#endif // SHARES_H
```

4.2 Share.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>

using namespace std;

Shares::Shares()
{
   company = "";
   nrShares = 0;
   unitValue = 0.0;
   for (int i = 0; i < 5; i++)
   {
   pastUnitValues[i]=0;
   }
}

Shares::Shares(string pCompany, int pNrShares, float pShareValue)
{
  company = pCompany;
   nrShares = pNrShares;
   unitValue = pShareValue;
   for (int i = 0; i < 5; i++)
   {
   pastUnitValues[i]=0;
   }


}

Shares::~Shares()
{

}

string Shares::getCompany()
{
   return company;
}

int Shares::getnrShares()
{
 return nrShares;
}

float Shares::getunitValue()
{
   return unitValue;
}
```

```
bool operator==(const Shares &s1, const Shares &s2)
{
   if (s1.company==s2.company)
      return true;
   else
      return false;
}

void Shares::setUnitValue(float newUnitValue)
{
 unitValue = newUnitValue;
}

void Shares::setShares(int newShares)
{
   nrShares=newShares;
}

double Shares::getPastUnitValues(int whichOne)//i
{
 return pastUnitValues[whichOne];//1
}

void Shares::setPastUnitValues(float UnitValue,int whichOne)
{
   pastUnitValues[whichOne] = UnitValue;
}

void Shares::updateShares(Shares & s1)//aShare.updateShares(NewObj);
{
 int oldShares = getnrShares();
 int newShares = s1.getnrShares();
 int totalShares = oldShares + newShares;
 s1.setShares(totalShares);

 for (int i = 0; i < 5; i++)
 {
 if (i<4)
 {
 s1.setPastUnitValues(s1.getPastUnitValues(i+1),i);//1
 }
 s1.setPastUnitValues(s1.getunitValue(),4);//void Shares::setPastUnitValues(float UnitValue,int whichOne)

}

}

istream& operator >>(istream& ins, Shares &s)
{
   ins >> s.company;
   ins >> s.nrShares;
```

```
   ins >> s.unitValue;
   for (int i = 0; i < 5; i++)
   {
   ins >> s.pastUnitValues[i];

   }
   return ins;
}

ostream& operator << (ostream& outs, Shares & s)
{
 outs <<"Company " << s.company;
 outs <<"Number of Shares: " << s.nrShares;
 outs << "Unit Value: " << s.unitValue;
 outs << "Past Unit Values: " << endl;
 for (int i = 0; i < 5; i++)
 {
    outs << s.pastUnitValues[i];
 }
 return outs;
}
```

4.3 Main()

```
#ifndef SHARES_H
#define SHARES_H
#include <iostream>
#include "share.h"
#include <string>
#include <fstream>
#include <cstdlib>

using namespace std;


class Shares
{
   public:
      Shares();
      Shares(string pCompany, int pNrShares, float pShareValue);
      ~Shares();
      string getCompany();
      void setShares(int NewShares);
      void setUnitValue(float NewUnitValue);
      void setPastUnitValues(float UnitValue,int whichOne);
      double getPastUnitValues(int whichOne);
      int getnrShares();
      float getunitValue();

      friend bool operator==(const Shares &s1, const Shares &s2);
      void updateShares(Shares & s1);
      friend istream & operator >> (istream& ins, Shares & s);
      friend ostream& operator << (ostream& outs, Shares & s);
```

```
  protected:

  private:
    string company;
    int nrShares;
    float unitValue;
    double pastUnitValues[5];
};

#endif // SHARES_Hdefine product class bank account
```

4.4 Derive SavingsAccount (NOT SURE WHAT THEY MEAN. The header file should contain compiler directives to prevent multiple definitions:

4.5 Implement Overloaded constructor for the class SavingsAccount

```
SavingsAccount::SavingsAccount(int accNr, double initBalance, double ir) : BankAccount(accNr,
initBalance),interestRate(ir){};
```

4.6 Member function postInterestRate() adapt BankAccount

```
protected:
  double balance;
```

We protect balance so that the class that is Derived from BankAccount can access the variable. We can't access it if its private.

5 TEMPLATES

5.1 Interface for Class Template

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


using namespace std;

template <class T>

class Data
{
Data();
public:
T calcAverage();
void captureData(T theData[]);
T findMin();
T findMax();
private:
vector <T> theData;
};
```

5.2 Implementation of calcAverage

```
template <class T>
T Data<T>::calcAverage()
{
   T total = 0.0;
   for (int i = 0; i < theData.size(); i++)
   {
      total = total+theData[i];
   }
return total/theData.size();
}
```

5.3 Provide a declaration for a Data object intended to contain object of class Prices (NOT SURE)

```
Data < double>  prices;
```