NAME: MODEL ANSWERS

STUDENT NUMBER: . . . .

Question 1

Assume that you are coding in C++ before the advent of the Standard Template Library. You decide to design a class called `vector` for storing arrays of elements of type `double`.

Your class must provide a constructor that creates space for a user specified number of doubles on the heap and initializes each of them to zero. You must be able to index a vector as you would an array and a vector must be able to grow at run time.

Study the header for `vector` given below. Some of the members of `vector` have completed code. Some members are incomplete.

```
class vector {

    int sz;     // the number of elements (the size)
    double* elem;   // pointer to the first element
    int space;      // size + free_space

    public:
        vector();   // constructor
        vector(int s);  // constructor
        vector(const vector&) ; // copy constructor:
        ~vector(){ delete[ ] elem; };  // destructor
        int size() const { return sz; };    // get the current size
        double get(int n) { return elem[n]; };  // access: read
        void set(int n, double v) { elem[n]=v; };   // access: write
        void reserve(int newalloc); // get more space
        int capacity() const { return space; }  // get available space
        void resize(int newsize);   // grow (or shrink)
        vector& operator=(const vector& a); // copy assignment
        double& operator[ ](int n) { return elem[n]; };  // access:
        void push_back(double d);   // add element

};
```

Provide the C++ code for the incomplete members of `vector` identified on the following pages.

a) give suitable code for both constructors. [4]

```
vector::vector()
    :sz(0),space(8), elem(new double[8])

{


}



vector::vector(int s)
    :sz(s),space(s), elem(new double[s])

{

    for (int i = 0; i<sz; ++i)
        elem[i] = 0.0;

}
```

b) give suitable code for the **copy** constructor.
hint: the copy constructor is invoked using: `vector v2 = v1;` [3]

```
vector::vector(const vector& a)

    :sz(a.sz),  elem(new double[a.space], space(a.space))

    // initialize sz elements (by copying)

{

    for (int i = 0; i<sz; ++i)
        elem[i] = a.elem[i];

}
```

c) give suitable code for the `reserve` member. [6]

```
void vector::reserve(int newalloc)
// make space for newalloc elements

{

    if (newalloc<=space) return;
    // never decrease allocation

    double* p = new double[newalloc];
    // allocate new space

    for (int i=0; i<sz; ++i) p[i]=elem[i];
    // copy old elements

    delete[ ] elem;
    // deallocate old space

    elem = p;

    space = newalloc;

}
```

d) give suitable code for the `resize` member. [3]

```
void vector::resize(int newsize)

    // make the vector have newsize elements
    // initialize each new element 0.0

{
    reserve(newsize);
    // make sure we have sufficient space

    for(int i = sz; i<newsize; ++i) elem[i] = 0;
    // initialize new elements

    sz = newsize;

}
```

e) give suitable code for the assignment operator, =.

hint: if you do not currently have enough space for the copy you must allocate new space on the heap and clean up after the copy. [10]

```
vector& vector::operator=(const vector& a) {

    // self-assignment, no work needed
    if (this==&a) return *this;

    // enough space, no need for new allocation
    // copy elements and ajust size.
    if (a.space <= space) {
        for (int i = 0; i<a.sz; ++i)
            elem[i] = a.elem[i];
        sz = a.sz;
        return *this;
    }

    // not enough space
    // make space, copy and cleanup
    double* p = new double[a.space];
    for (int i = 0; i<a.sz; ++i) p[i] = a.elem[i];
    delete[ ] elem;
    sz = a.sz;
    space = a.space;
    elem = p;
    return *this;
}
```

f) give suitable code for the `pushback` member.
hint: if your vector is full reserve more space before inserting new element. [4]

```
void vector::push_back(double d)

    // increase vector size by one
    // initialize the new element with d

{

    // no space: grab some
    if (sz==0)
        reserve(8);

    // no more free space: get more space
    else if (sz==space)
        reserve(2*space);

    // add d at end
    elem[sz] = d;

    // and increase the size (sz is the number of elements)
    ++sz;

}
```

Question 2

Show how to generalize your `vector` of `double` class by writing down the C++ header file for a `template` version of `vector`.

ie: show how to *templatise* the header file given at the start of question 1. . [10]

```
template<class T> class vector {

    int sz;      // the size
    T* elem;         // pointer to the elements
    int space;       // size+free_space

public:
    // default constructor
    vector() : sz(0), elem(0), space(0);

    // explicit constructor
    explicit vector(int s)
        :sz(s), elem(new T[s]), space(s) {}

    // copy constructor
    vector(const vector&);

    // copy assignment
    vector& operator=(const vector&);


    // destructor
    ~vector() { delete[ ] elem; }

    // get and set
    T get(int n){ return elem[n]; }
    void set(int n, T v) { elem[n]=v; }

    // access: return reference
    T& operator[ ] (int n) { return elem[n]; }

    // the current size
    int size() const { return sz; }

    // add new element
    void push_back(T d);

};
```

Question 3

Construct a C++ function that, given a `string` representing a valid `filePath` to a text, makes use of `vector<char>` from the Standard Template Library to compute and return the most commonly used alphabetic character in the text.

(btw: this is usually 'e' in English texts). [10]

```
char commonestChar(string filePath) {

    vector<char> chars;

// load all alpha chars into a vector<char>
    char ch;
    ifstream ifs(inFileName.c_str());
    while(ifs.get(ch)){
        ch = tolower(ch);
        if (ch >= 'a' && ch <= 'z')
            chars.push_back(ch);
    };

// now sort the chars so that we can count each type of char
    sort(chars.begin(),chars.end());
// add one more char so that we can count last character
    chars.push_back('Z');

// now find the commonest char
    int count = 0;
    int bestCount =0;
    char bestChar = 'a';
    for (int i=1; i<chars.size(); ++i) {
            if(chars[i-1]!=chars[i]) {
                if (count > bestCount) {
                    bestCount = count;
                    bestChar = chars[i-1];
                    count = 0;
                };
            };
            ++count;
    }

// and return it.
    return bestChar;

}
```