Project_1

1. Create a class – template my_vect:

```
template <class T> class my_vect
{
      T *dat;        //pointer to the array of T type
      size_t ndim;   // number of items for which memory was dynamically allocated
      size_t last;   // an index that points to the first "empty" element of the
                     //array

public:
      my_vect(size_t dim);  // allocates memory for the array dat to dim
                            // elements when creating an object
      ~my_vect();           //release memory occupied by array dat.

      //class methods for searching   Find(….)
      T *get_begin() { return dat; }   //returns pointer to dat[0]
      T *get_end()   { return &dat[last]; }

      void push(const T &ob); // push object T in position last of array dat
                              //(to dat[last]) and reset last to point
                              //at the first free position
      T * pop();        //returns the last element from dat array and reset last
                         //to the previous element.
      void insert(const T ob[], size_t ind, size_t numb);  // inserts an array ob
                         //in the array dat immediately after the element
                         //dat[ind]; numb - number of elements in the array ob.
      void erase(const T *ob); //remove the * ob element from the array dat and
                              //shifts the array elements so that when removed,
                              //the array elements are placed contiguously.


private:
      void realloc();   //if last >= ndim – increases ndim and reallocates
                         //memory for array dat.};

};
```

Add the method:

- clear_all – removal of all array elements
- Overload the <<, >> operators for writing and reading array dat into a binary file.
- Overload the operator [ ] to get and assign an element of the array dat[ind].

2. Create the template-function Find:

```
template <class T, class Key>
T * find(const T *p_begin, const T *p_end, const Key &k);
```

p_begin – pointer to the first element of the array tab from which the search begins; p_end – the first element of the array tab which is after of the last element from the range of search; Key k – search criteria (for given example – the vertex number). Returns pointer to the found object or NULL in the case of unsuccessful search.

3. Create class mcoord, representing the coordinates of the vertex on the plain:

```
class mcoord
{
protected:
      double *pcoord;  //pcoord[0] – coordinate x, pcoord[1] – coordinate y
public:
      mcoord(double xx, double yy);
      mcoord() {pcoord = NULL; }
      ~mcoord() { ……………… }
};
```

Create a class node that inherits class mcoord.

```
class node : public mcoord
{
      int numb;        //vertex’s number
      char str[128];  //vertex’s name (for example, A or vertex A)
public:
      node(int nb, char *st, double xx, double yy);  //parameterized constructor
      node();
      ………………….
};
```

For the node class, overload the operator =, add a copy constructor, overload the operator == (for the correct operation of the Find (...) function; overload the <<, >> operators for inserting an object into any stream and retrieving from any stream.

4. Create a system for handling errors, warnings and messages. All errors, warnings and messages must be placed in one file, not spread over the entire code.
5. The dynamic allocation/release of memory is made by the operators new / delete
6. Create an interface on the basis of an infinite while () loop, which can be broken by introducing some code from the monitor. The interface should contain:
   - Add an object.
   - Delete an object
   - Delete all
   - Modify an object
   - Insert an array of objects
   - Find all objects (by the vertex number – possible, several objects can have the same number)
   - Save data to a binary file.
   - Read data from a binary file.
   - View data on the monitor
   - Quit.
7. Put each class, error (message) handling and file containing main () functions into separate * .cpp files; * .h.
8. Present the project in electronic form as a project archive together with the data file.