```cpp
#include <iostream>
#include <string>

class StorageException : public std::runtime_error
{
public:
  StorageException(): std::runtime_error("Out of the boundary"){}
};
```

```cpp
template <class T>
class Storage
{
private:
  T *array;
  int size;
public:
  /**
   * this function is the default constructor
   * @param length is the length of the array
   */
  Storage(int length);
  /**
   * This functionm is the copy constructor
   * @param elem is the object that pass to the copy constructor
   */
  Storage(const Storage &elem);
  /**
   * This is default constructor
   */
  ~Storage();
  /**
   * This function overload the [] symbol
   * @param index is the place on array
   * @return the place on the array
   */
  T &operator[](const int &index);
  /**
   * This function is to overload the << symbol so that it can directly output the object
   * @tparam CT is the template type
   * @param stream_insertion is the ostream object used to output
   * @param obj is the object that output to
   * @return the output value
   */
```

```
  template<class CT>
  friend std::ostream &operator<<(std::ostream &stream_insertion,  Storage<CT> &obj);
  /**
   * This function is to determine the maximum
   * @tparam DT is the template type
   * @param elem is the elements in the array to decide the maximum
   * @return
   */
  template<class DT>
  friend DT maximum(Storage<DT> &elem);
  /**
   * This function is to search things in the array
   * @tparam ET is the template type
   * @param searchVal is the value to search
   * @param elem is the array that search for
   * @return the boolean result of whether found or not
   */
  template <class ET>
  friend bool searchElement(ET searchVal, Storage<ET> &elem);
};
template <class T>
T &Storage<T>::operator[](const int &index)
{
  if (index >= size || index < 0)
  {
    throw StorageException();
  }
  return array[index];
}

template<class PT>
Storage<PT>::Storage(int length)
{
  size = length;
  if(size <= 0)
  {
   throw StorageException();
  }
  array = new PT [size];
}
```

```cpp
template<class T>
Storage<T>::Storage(const Storage &elem)
{
  size = elem.size;
  if(size <= 0)
  {
    throw StorageException();
  }
  // allocate memory
  array = new T [size];
  for(int count = 0; count < size; count++)
  {
    *(array + count) = *(elem.aptr + count);
  }
}
```

```cpp
template<class T>
Storage<T>::~Storage()
{
  // release all the allocated memory
  delete[] array;
  std::cout << "Deleting all the array elements ... " << std::endl;
}
```

```cpp
template<class CT>
std::ostream &operator<<(std::ostream &stream_insertion, Storage<CT> &obj)
{
  for (size_t i = 0; i < obj.size; i++)
  {
    stream_insertion << obj[i];
    if (i == obj.size - 1)
    {
      continue;
    }
    stream_insertion << ", ";
  }
  return stream_insertion;
}
```

```cpp
template<class DT>
DT maximum(Storage<DT> &elem)
{
  DT largest;
 // DT read;
  if(elem.size == 0)
  {
    throw StorageException();
  }
  for(unsigned looptimes = 0; looptimes < elem.size; looptimes++)
  {
    if(elem.array[looptimes] > largest)
    {
      largest = elem.array[looptimes];
    }
  }
  return largest;
}
```

```cpp
template<class ET>
bool searchElement(ET searchVal, Storage<ET> &elem)
{
  if(elem.size == 0)
  {
    throw StorageException();
  }
  for (size_t count = 0; count ⩽ elem.size; count++)
  {
    if (elem.array[count] == searchVal)
    {
      return true;
    }
  }
  return false;
}
```

```cpp
int main()
{
  try
  {
    //Construct object using the size as parameter to the array size
    Storage<int> myIntStorage( length: 4);
    Storage<double> myDoubleStorage( length: 4);
    Storage<std::string> myStringStorage( length: 4);

    //fill up the array
    for (int count = 0; count < 4; count++)
    {
      myIntStorage[count] = count+1;
      myDoubleStorage[count] = count * 2.14;
    }
    myStringStorage[0] = "Dog";
    myStringStorage[1] = "Cat";
    myStringStorage[2] = "AAA";
    myStringStorage[3] = "BBB";

    // Display the values in the SimpleVectors.
    std::cout << "Here is the int array elements:  " << myIntStorage << std::endl;
    std::cout << "Here is the double elements:  " << myDoubleStorage << std::endl;
    std::cout << "Here is the string elements: " << myStringStorage << std::endl;


    //Display the max element of the array
    int maxIntElement = maximum( &: myIntStorage);
    std::cout << std::endl <<  "Here is the max int element in the array: " << maxIntElement << std::endl;
    double maxDoubleElement = maximum( &: myDoubleStorage);
    std::cout << "Here is the max double element in the array: " << maxDoubleElement << std::endl;
    std::string maxStringElement = maximum( &: myStringStorage);
    std::cout << "Here is the max string element in the array: " << maxStringElement << std::endl;

    //Search elements in the array
    int searchIntValue = 4;
    bool returnFlag_Int = searchElement(searchIntValue,  &: myIntStorage);
    std::cout << std::endl << "The result of finding is: " << returnFlag_Int << std::endl;
    double searchDoubleValue = 6.42;
    bool returnFlag_Double = searchElement(searchDoubleValue,  &: myDoubleStorage);
    std::cout << "The result of finding is: " << returnFlag_Double << std::endl;
    std::string searchStringValue = "Dog";
    bool returnFlag_String = searchElement(searchStringValue,  &: myStringStorage);
    std::cout << "The result of fining is: " << returnFlag_String << std:: endl << std::endl;

  } catch(const char *error)
  {
    std::cout << "Error: " << error << std::endl;
  }

  return 0;
}
```

Normal Running result:

```
Lab5_TemplateException ×
/Users/AKIRA/ComputerScienceRelated/Spring2021-CS181/Lab5-TemplateException/cmake-build-debug/Lab5_TemplateException
Here is the int array elements:  1, 2, 3, 4
Here is the double elements:  0, 2.14, 4.28, 6.42
Here is the string elements: Dog, Cat, AAA, BBB

Here is the max int element in the array: 4
Here is the max double element in the array: 6.42
Here is the max string element in the array: Dog

The result of finding is: 1
The result of finding is: 1
The result of fining is: 1

Deleting all the array elements ...
Deleting all the array elements ...
Deleting all the array elements ...

Process finished with exit code 0
```

Error Showing
7(A):

```
//Construct object using the size as parameter to the array size
Storage<int> myIntStorage( length: 4);
Storage<double> myDoubleStorage( length: 4);
Storage<std::string> myStringStorage( length: 2);

//fill up the array
for (int count = 0; count < 4; count++)
{
    myIntStorage[count] = count+1;
    myDoubleStorage[count] = count * 2.14;
}
myStringStorage[0] = "Dog";
myStringStorage[1] = "Cat";
myStringStorage[2] = "AAA";
myStringStorage[3] = "BBB";
/Users/AKIRA/ComputerScienceRelated/Spring2021-CS181/Lab5-TemplateException/cmake-build-debug/Lab5_TemplateException
libc++abi.dylib: terminating with uncaught exception of type StorageException: Out of the boundary

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

7(B):

```
//Construct object using the size as parameter to the array size
Storage<int> myIntStorage( length: 0);
Storage<double> myDoubleStorage( length: 0);
Storage<std::string> myStringStorage( length: 0);
```

```
/Users/AKIRA/ComputerScienceRelated/Spring2021-CS181/Lab5-TemplateException/cmake-build-debug/Lab5_TemplateException
libc++abi.dylib: terminating with uncaught exception of type StorageException: Out of the boundary

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

7(C)

```
    bool returnFlag_String = searchElement(searchStringValue, &: myStringStorage);
    std::cout << "The result of fining is: " << returnFlag_String << std:: endl << std::endl;

} catch(const char *error)
{
    std::cout << "Error: " << error << std::endl;
}

return 0;
}
```