Name: ____________________ ID: __________

Object-Oriented Programming Lab #6 **Feb 24th, 2023**

Managing Memory

1. Write functions and programs that modifies values through pointers, verify correctness of all functions with test programs and ensure that all programs run as expected.

1.1) Modify the following program using **`sizeof`** operator to determined how many bytes are there in an int, a double, a bool and in variables in the program. Guess for answers about how **`sizeof`** calculates the number of bytes for different types. Check if the result match your expectation.

```cpp
#include <vector>
#include <array>

int main()
{
    double coords[3] = {};
    double* p1 = coords;

    std::array<double, 4> pt4d;
    auto arr_it = pt4d.begin();

    std::vector<double> vec;
    auto vec_it = vec.begin();
}
```

You are expected to answer the following questions:

- Why the value of **`sizeof(coords)`** and **`sizeof(p1)`** are different?
- Can we compute the number of elements in **`coords`** using **`sizeof`**? If so, how do we write the computation?
- Can we compute the number of elements in **`coords`** through **`p1`**? If so, how do we write the computation?
- Can we compute the number of elements in **`vec`** using **`sizeof`**? If so, how do we write the computation?

1.2) Write a function, **`void to_upper(char* s)`**, that replaces all lowercase characters in the C-style string **`s`** with their lowercase equivalents. **Do not use** any standard library functions. A C-style string is a zero-terminated array of characters, so if you find a char with the value 0 you are at the end (**stop reading** a char at that point).
For example, **`"Hello, World!"`** becomes **`"HELLO, WORLD!"`**.

1.3) Write a function, **`char* rev_dup(const char* s)`**, that copies a C-style string into memory it allocates on the free store in **reverse order** from the original (and retains 0 at the end of the string). **Do not use** any standard library functions.
For example, duplicating **`"Hello, X"`** reversely will create a C-style string **`"X ,olleH"`**.

1.4) Write a function, **`char* read_text(std::istream&)`**, that reads characters from an input stream into a char array that you allocate on the free store. Read individual characters until an exclamation mark (**`!`**) is entered. **Do not use** any standard library functions. Do not worry about memory exhaustion.

For example, when entering **`"Hello, X!, Y!, Z"`** from the input, a C-style string **`"Hello, X"`** is created from the function.

Advice: Use cin.get() to read a character without skipping whitespaces.

1.1)	☐	**1.2)**	☐	**1.3)**	☐	**1.4)**	☐

2. Write a class for representing an N-dimension vector **`Nd_vec`** (not the standard library vector), **without using the C++ standard library container** and **use the free store memory** to store the data, along with basic operations and test programs.

2.1) Write an **`Nd_vec`** class which stores an N-dimension of number components.

You should start with the following skeleton for class **`Nd_vec`**:

```
class Nd_vec {
public:
    // copy constructor
    Nd_vec(/* ... */) = delete;

    // assignment operator
    /* ... */ operator=(/* ... */) = delete;

    // other operations
private:
    /* Data Impl. Type */ data;
};
```

You are required to:

- Provide appropriate **constructors** for class **`Nd_vec`**
- Provide an appropriate **destructor** for class **`Nd_vec`**
- Provide a **member function**, **`dimension()`**, for getting the dimension of an **`Nd_vec`** object
- Complete declarations for the **copy constructor** and the **assignment operator** in the skeleton above
- Write a test program for testing all use cases of a **`Nd_vec`** object and its operations including the test for constructing **`Nd_vec`** object and getting its dimension
- Verify that you free the memory correctly in the **destructor** of the class

2.2) Modify the code from **2.1)** by adding a member function `print` to print the contents of an `Nd_vec` to the output stream. Write a test program for testing all use cases of the `print` function.

2.3) Modify the code from **2.2)** by adding the following operations to `Nd_vec`:

- **copy constructor** for class `Nd_vec`
- **assignment operator** for class `Nd_vec`

Write a test program for testing all use cases of a `Nd_vec` object and its operations including the test for passing `Nd_vec` to a function, returning `Nd_vec` from a function, constructing a `Nd_vec` from another `Nd_vec`, and copying a `Nd_vec` object.

2.4) Modify the code from **2.3)** add the following operations:

- Member function, `clear()`, for deallocating all free store memory used by `Nd_vec` object
    - After calling `vec.clear()` for the `Nd_vec` object `vec`, its dimension should be zero and the object would contains no data for its contents
- `add(v1, v2)` for creating a new `Nd_vec` by **adding** two vectors `v1` and `v2`
- `subtract(v1, v2)` for creating a new `Nd_vec` by **subtracting** `v1` with `v2`
- `scale(v, x)` for creating a new `Nd_vec` by **scaling** a vector `v` by `x`

Add additional support operations as needed. Write a test program for testing all of the above operations.

2.1)	☐	**2.2)**	☐	**2.3)**	☐	**2.4)**	☐

3. Write a class for representing an ASCII picture, **without using the C++ standard library container** and **use the free store memory** to store the data, along with basic operations and test programs.

3.1) Write a `Picture` class which stores a collection of rows of a text string for its content. The longest row determines the width and the number of rows represents the height.
You are required to:

- Provide appropriate **constructors** for class `Picture`
- Provide an appropriate **destructor** for class `Picture`
- Provide appropriate **copy constructor** for class `Picture`
- Provide appropriate **assignment operator** for class `Picture`
- Provide appropriate **member functions** for getting the width and the height of a `Picture` object
- Provide a **member function**, `print` to print the contents of a picture to the output stream
- Write a test program for testing all use cases of a `Picture` object and its operations including the test for constructing `Picture` object, getting its width and height, printing its contents, passing `Picture` to a function, returning `Picture` from a function, constructing a `Picture` from another `Picture`, and copying a `Picture` object.
- Verify that you free the memory correctly in the **destructor** of the class.

3.2) Modify the code from **3.1)** add the following operations:

- Member function, `clear()`, for deallocating all free store memory used by `Picture` object
  - After calling `pic.clear()` for the `Picture` object `pic`, its width and height should be zero and the object would contains no data for its contents
- `hcat` for creating a new picture by **concatenating** two pictures **horizontally**
- `vcat` for creating a new picture by **concatenating** two pictures **vertically**

Add additional support operations as needed. Write a test program for testing all of the above operations.

3.1) ☐	**3.2)** ☐