<Project Name>

Design Document

Authors:

<person>

<person>

———————————————————————————————————————

DO NOT INCLUDE CONTENT BETWEEN THE LINES IN YOUR FINAL DRAFT

**The point of this assignment is to give you a guideline when actually writing your program. Remember:**

* Be clear and concise. I do not need a 100 page document- if your work would lose your attention because of its length, then it’s too long. **You want enough detail so that this program could be written by anyone with a knowledge of C++ and OOP,** but not so long that people stop reading.
* Pay attention to details. You want your design to be complete and easy to replicate. A good rule of thumb is to use the peanut butter & jelly direction rule- break everything down into easy to understand items so that there isn’t accidental misinterpretation.
* Make sure your writing is readable, and the tone and language are consistent.
* This is a *very* modified design document template, with portions from typical design documents excluded. This is to allow you to focus on the program design aspect.
* The supplied headers should not be changed, and the general formatting (ie, indenting/bolding) should be similar.
* Anything that is italicized is directions for that section- **they should not be in the final document, and neither should these directions between the dotted lines.**
* Any time you see <directions>, you should provide the requested information **without the angled brackets.** For example, my document author would be Sara Davis, NOT <Sara Davis>

———————————————————————————————————————

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description of changes | Author |
| 7/7/2023 | 0.0 | Initial draft | Sara Davis |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. **Introduction**
   1. *What is the programming problem?*
   2. *Who is the intended audience?*
   3. *What problems are you trying to solve?*

**1.1 Design Objectives**

* 1. *What functionality should the program hide from the end user?*
  2. *What explicitly does the end user need to be able to access?*
  3. *What is the overall scope for your design; in other words, describe all of the general features and functions that you will need here. You don’t need to give explicit detail, but you do need to explain what the major functionality of the program is.*

**2. Design Overview**

* 1. *This section normally describes the operating system (OS) used, architectures, etc. I only need the subfields below addressed (in other words, there shouldn’t be text between header 2 and subheader 2.1)*

**2.1 Constraints and Assumptions**

* 1. *Describe all major design constraints*
     1. *What* ***has to be true*** *for the program to work as intended?*
     2. *What constraints were given in the readme for the assignment?*
  2. *If a constraint is placed by a design choice, then mention any other options you considered, their benefits/downsides, and why you selected the option you chose to solve the problem. For example:*

*The use of no graphical interface for the battleship class restricts the program to displaying plain text. The program could use a display method specific to the class, which would provide the benefit of easy access to display at the expense of data hiding, or it could simply user class getters to display the number and class. The display method was chosen because it provides direct access to an implementation that does not necessarily need to be hidden.*

*3***. Structural Design**

* 1. *This section normally describes how classes and functions interact.There should not be any text between header 3 and subheader 3.1*

**3.1 Design Explanation and Rationale**

* 1. *Explain how you chose to design the program. Use this section to justify each class you will build and any atypical functions (not constructors/getters/setters)*
     1. *What classes were used?*
     2. *How do they interact?*
     3. *Why did you choose those classes and those interactions?*

**3.2 Class Diagram**

* 1. *Provide a UML diagram for the program you described in 3.1*
     1. *I use draw.io, but you can use whatever you want including powerpoint and keynote*
     2. *Every class should be shown in the diagram, and the interactions should be conveyed through UML notation.*
     3. *You do not need to include your constructor/getter/setter methods as part of the UML diagram, only methods that are out of the ordinary.*
     4. *The digram should have a figure caption that describes, in words, how the classes interact. There should be a figure number associated with the diagram, and you should refer back to that figure number as necessary in your writing.*
     5. *I have* ***not*** *included a class diagram for DynamicArray- you will need to create one yourself and add it to the global class diagram* ***if you choose to reuse your DynamicArray class from assignment 3****. Template classes are indicated by using a dotted line box that holds the template and its possible dependent values.*

**3.3 Class Descriptions**

* 1. *In this section, describe each class, its attributes, and its methods* ***in detail.*** *The UML gave us the general view and description, this is where you provide detail.*
  2. *Each class should be listed as a subsection under section 3.3, as shown below for DynamicArray.*
     1. *If you are not using DynamicArray as part of your design, then you can remove the description below- but it’s still a good example of expectations*
  3. *Each subsection should describe the point of the class, any constraints (such as not being able to create an object out of an ABC), and list any attributes/methods of the class from the UML in 3.2.*
  4. *Each class subsection should give the* ***name, data type, description, and constraints for each property in the class.***
  5. *Each class subsection should give the* ***name, return type and value, parameters, purpose, and a description of what is done*** *for every method in the class.*

**3.3.1 Class: DynamicArray**

**Purpose**: Dynamically allocate an array that can be resized to fit any number of items, regardless of data type.

**Constraints**: Template class. Must be instantiated with data type in angled brackets.

**3.3.1.1 Attribute Descriptions**

* + - * 1. **Attribute**: array

**Type**: pointer of **general** data type decided by template

**Description**: Stores items of data type decided by template

**Constraints**: Must use the template type (no explicit data typing). Memory should be allocated in class methods (see later)

* + - * 1. **Attribute**: maxSize

**Type**: int

**Description**: maximum size of the array; updated

**Constraints**: only updated at instantiation and when array is resized

* + - * 1. **Attribute**: currentSize

**Type**: int

**Description**: the current number of items that are in the array; this may or may not be different than the maxSize.

**Constraints**: only updated at instantiation and when array has item added.

**3.3.1.1 Method Descriptions**

* + - * 1. **Method**: DynamicArray()

**Return** **Type**: DynamicArray object (implicit)

**Parameters**: none

**Return** **Value**: Current instance of object (implicit)

**Pre**-**Condition**: Memory not assigned to object

**Post**-**Condition**: Memory assigned to object

**Attributes read/used**: array, maxSize, currentSize

**Description**: Sets initial maxSize to 3 and currentSize to 0; assigns memory to array using maxSize.

**Methods** **Called**: None

* + - * 1. **Method**: DynamicArray(int m, int c, T a)

**Return** **Type**: DynamicArray object (implicit)

**Parameters**: m - the current max size of the array; c- the current size of the array; a- the array pointer of type template to be copied from

**Return** **Value**: Current instance of object (implicit)

**Pre**-**Condition**: Memory not assigned to object, max/current/array contents known at instantiation.

**Post**-**Condition**: Memory assigned to object

**Attributes read/used:** array, maxSize, currentSize

**Description**: Sets initial maxSize to m and currentSize to c; assigns memory to array using maxSize; iterates through passed array and copies all contents from the array parameter into the array attribute.

**Methods Called**: None

` 3. **Method**: DynamicArray(const DynamicArray& d)

**Return** **Type**: DynamicArray object (implicit)

**Parameters**: DynamicArray to copy from

**Return** **Value**: Current instance of object (implicit)

**Pre**-**Condition**: Memory not assigned to object; new object instantiated with = sign

**Post**-**Condition**: Memory assigned to object

**Attributes** **read**/**used**: array, maxSize, currentSize

**Description**: copies the maxSize, currentSize from an object of type DynamicArray at instantiation. Allocates memory to array, iterates through d’s array and copies all contents from d’s array to array.

**Methods** **Called**: None

4. **Method**: ~DynamicArray()

**Return** **Type**: None

**Parameters**: None

**Return** **Value**: Current Object with no assigned memory

**Pre**-**Condition**: Object exists and has been allocated memory

**Post**-**Condition**: Memory assigned to object is freed

**Attributes read/used**: array, currentSize

**Description**: clears all memory from array, updates currentSize to 0

**Methods** **Called**: None

5. **Method**: operator =(const DynamicArray& d)

**Return** **Type**: instance of current object

**Parameters**: DynamicArray to copy from

**Return** **Value**: Current instance of object using this keyword (dereferenced)

**Pre**-**Condition**: DynamicArray object is instantiated, assignment requested after instantiation.

**Post**-**Condition**: Contents of d stored as deep copy in current

**Attributes read/used**: array, maxSize, currentSize

**Description**: copies the maxSize, currentSize from an object of type DynamicArray at instantiation. Allocates memory to array, iterates through d’s array and copies all contents from d’s array to array.

**Methods** **Called**: None

6. **Method**: getMaxSize

**Return** **Type**: int

**Parameters**: none

**Return** **Value**: The current maxSize

**Pre-Condition**: getMaxSize has been set

**Post-Condition**: current max size is returned

**Attributes read/used**: maxSize

**Description**: Returns the value stored in maxSize

**Methods Called**: None

7. **Method**: getCurrentSize

**Return Type**: int

**Parameters:** none

**Return Value**: The current number of items stored in array

**Pre-Condition**: items have been stored in array

**Post-Condition**: current size is returned

**Attributes read/used**: currentSize

**Description**: Returns the value stored in currentSize

**Methods Called**: None

8. **Method**: getArray

**Return Type**: a pointer of **general** datatype determined by template

**Parameters**: none

**Return Value**: The array

**Pre-Condition**: Array has been initialized with memory

**Post-Condition**: array is returned

**Attributes read/used**: array

**Description**: Returns a pointer to the array

**Methods Called**: None

9. **Method**: addItemToArray

**Return Type**: none

**Parameters**: An item of **general** type decided by template

**Return Value**: none

**Pre-Condition:** Array has had up to maxSize number of items inserted

**Post-Condition**: Item is added to array & currentSize is updated, maxSize is updated if necessary

**Attributes read/used:** array, maxSize, currentSize

**Description**: Checks if maxSize and currentSize are the same. If they are, array is resized. Regardless of resizing, item is added after the last added item and currentSize is incremented. Replaces array setter.

**Methods Called**: resizeArray (if necessary)

10. **Method**: resizeArray

**Return Type:** none

**Parameters:** none

**Return Value:** none

**Pre-Condition:** maxSize of array == currentSize of array

**Post-Condition:** maxSize is larger than currentSize

**Attributes read/used**: array, currentSize, maxSize

**Description**: maxSize is doubled; a temporary array is the new maxSize worth of memory. Each item from the array property is copied into the temporary array. The array property’s memory is freed. The array pointer is set to the temporary array’s pointer. Should be **private**.

**Methods Called**: None

11. **Method**: getElement(int ind)

**Return Type**: **general** type decided by template

**Parameters**: index- the index of the array accessed

**Return Value**: the item stored in array at index

**Pre-Condition**: array has contents

**Post-Condition**: specific index contents gathered

**Attributes read/used:** array, currentSize

**Description**: checks if requested index is less than or equal to currentSize. If it is not, outputs an error. If it is, the element of array at that index is returned.

**Methods Called**: None

11. **Method**: clearArray()

**Return Type**: none

**Parameters**:none

**Return Value**: none

**Pre-Condition**: array has currentSize c, maxSize m, and array has memory

**Post-Condition**: array has currentSize 0, maxSize is 3, and array memory address is updated

**Attributes read/used**: array, currentSize, maxSize

**Description**: frees all of array’s memory, updates array maxSize to 3 and currentSize to 0, allocates new maxSize worth of memory to array

**Methods Called**: None

**3.4 Main and Helpers Descriptions**

* + 1. *Method* ***name, return type, parameters, return values, pre-condition state, post condition state, class objects read/used, method description, and call methods*** *should be supplied for main and every helper function (just like you did above for class methods).*
    2. *All helper functions should be placed in their own .h and .cpp file and main should be in main.cpp*
    3. *Basically, list any functions that you will need main to access so that you can get the classes to work together as intended. Ie, define the functions that will run the bulk of your program.* 
       1. *For example, we know that we will need to update the player board between rounds- we should define a helper function that will allow us to do that.*
       2. *Think about everything you want that function to do, and then decide what variables/objects should be created locally and what variables/objects should be passed as a parameter.*
    4. *This section will likely be >7 methods; it should map out all possible paths the program needs to take from main, through the method, and into each class and its use.*