Lab 05 - Array Storage

Instructions:

- **In a logic puzzle riddle game, several doors are presented such that each door has a riddle above it. Only one door leads to safety, and only one riddle is true. You must choose the correct door to escape by analyzing the riddles. In the header file '`LogicGame.h`', your objective is to create the game with 5 doors by constructing classes named *Slots*, which uses array storage, *Doors*, and *Game*,**
- **The header file must contain a header guard.**
- **The class must be defined within a namespace named '`dsl`'.**
- **The header file can only include the libraries *iostream*, *string*, *sstream*, *iomanip*, *cctype*, *cstdlib*, *ctime*, *stdexcept*, and '`Object.h`'.**
- **Define the special member function for each class such that the destructor is empty unless stated otherwise, and the copy constructor and assignment operator perform shallow copies.**
- **Each method, excluding special member functions, must include pseudocode as a comment above it to receive any credit.**
- **Your submissions must be submitted to the GitHub repository in the Lab05 directory.**
- **Cheating of any kind is prohibited and will not be tolerated.**
- **Violating or failing to follow any of the rules above will result in an automatic zero (0) for the lab.**

Grading

Task	Maximum Points	Points Earned
1	2.00	
2	1.00	
3	2.00	
Total	5.00	

Note: solutions will be provided for tasks colored blue only.

Task 1

- **Create the generic class *Slots* that publicly inherits *Object* and contains**
  - ☐ **a private generic array field named *slots* with a size of 5.**
  - ☐ **a private unsigned integer field named *size*.**
  - ☐ **a private string field named *name*.**
  - ☐ **a public default constructor that assigns default generic value to each element of *slots*, empty string to *name*, and 0 to *size*.**
  - ☐ **a public void method named `add()` that takes a constant generic reference parameter. It adds the generic parameter to the next available element of *slots* only if *slots* has available space.**
  - ☐ **a public void method named `before()` that takes two constant generic reference parameters. It adds the second parameter immediately before the first parameter only if the first parameter is present in *slots*. It overwrites the last element if *slots* is full.**
  - ☐ **a public void method named `after()` that takes two constant generic reference parameters. It adds the second parameter immediately after the first parameter only if the first parameter is present in *slots*. It overwrites the last element if *slots* is full.**
  - ☐ **a public void method named `remove()` that takes a constant generic reference parameter. It removes the parameter from *slots* if the parameter is present in slots.**
  - ☐ **a public void method named `exchange()` that takes two constant generic reference parameters. It replaces the first parameter with the second parameter in *slots* only if the first parameter is present in *slots*.**
  - ☐ **a public Boolean constant method named `contains()` that takes a constant generic reference parameter. It returns true if the parameter is in *slots*; otherwise, it returns false.**
  - ☐ **a public Boolean constant method named `empty()` that takes no parameters. It returns true if *slots* is empty; otherwise, it returns false.**
  - ☐ **a public Boolean constant method named `full()` that takes no parameters. It returns true if *slots* is full; otherwise, it returns false.**
  - ☐ **a public unsigned integer constant method named `occupants()` that takes no parameters and returns *size*.**
  - ☐ **a public string constant method named `label()` that takes no parameters and returns *name*.**
  - ☐ **a public void method named `label()` that takes constant string reference parameter and assigns the parameter to *name*.**
  - ☐ **a public constant generic reference constant method named `get()` that takes an integer parameter. It returns the element of *slots* whose index matches the parameter if the parameter is valid (in range [0,4]); otherwise, it throws an out-of-range error message.**
  - ☐ **a public overridden `toString()` method that returns a string that is a list occupied elements of *slots* such that each is on a separate line preceded by "*l i*: " where *l* is the value of *name* and *i* is the element's index.**

Task 2

- **Create the class *Doors* that publicly inherits *Object* and contains**
  - ☐ **a private string *Slots* field named *riddles*.**
  - ☐ **a private integer field named *answer*.**
  - ☐ **a public default constructor that assigns -1 to *answer* and "Door" to the *name* field of *riddles*.**
  - ☐ **a public void method named `add()` that takes a constant string reference parameter and invokes `add()` of *riddles* with the parameter as the argument.**
  - ☐ **a public void method named `before()` that takes two constant string reference parameters and invokes `before()` of *riddles* with the parameters as the arguments in order.**
  - ☐ **a public void method named `after()` that takes two constant string reference parameters and invokes `after()` of *riddles* with the parameters as the arguments in order.**
  - ☐ **a public void method named `remove()` that takes a constant string reference parameter and invokes `remove()` of *riddles* with the parameter as the argument.**
  - ☐ **a public void method named `exchange()` that takes two constant string reference parameters and invokes `exchange()` of *riddles* with the parameters as the arguments in order.**
  - ☐ **a public Boolean constant method named `contains()` that takes a constant string reference parameter and invokes `contains()` of *riddles* with the parameter as the argument.**
  - ☐ **a public Boolean constant method named `empty()` that takes no parameters and invokes `empty()` of *riddles*.**
  - ☐ **a public Boolean constant method named `full()` that takes no parameters and invokes `full()` of *riddles*.**

- ☐ **a public unsigned integer constant method named `labeled()` that takes no parameters and and invokes `occupants()` of *riddles*.**
- ☐ **a public void method named `set()` that takes integer parameter and assigns the parameter to *answer* only if the parameter is in the range [0,4].**
- ☐ **a public integer constant method named `solution()` that takes no parameter and returns *answer*.**
- ☐ **a public constant string reference constant method named `get()` that takes an integer parameter and invokes `get()` of *riddles* with the parameter as the argument.**
- ☐ **a public overridden `toString()` method that returns *riddles* as a string.**

Task 3

- **Create the class *Game* that publicly inherits *Object* and contains**
  - ☐ **a private readonly *Doors* pointer field named *game*.**
  - ☐ **a public default constructor that assigns null to *game*.**
  - ☐ **a public void method named `load()` that takes a readonly *Doors* pointer parameter and assigns the parameter to *game*.**
  - ☐ **a public Boolean method named `playable()` that takes no parameters. It returns true only if *game* is not null, full, and contains a solution; otherwise, it returns false.**
  - ☐ **a public void method named `play()` that takes no parameters. If playable, it displays *game* and then prompts the user to select a door. After receiving the user's input, it states the location of one incorrect door and then asks if the user wishes to change their answer. If the user inputs a y, it continually prompts the user to select another door until it differs from the original choice. Last, it states if the user was safe or not.**
  - ☐ **a public overridden `toString()` method that returns the string `"not configured"` if the game is not playable; otherwise, *game* as a string.**