

# Assignment 2: Overview

---

**Read the instructions carefully. Many things are different from HW1.**

Modify the provided skeleton code and complete the implementation of `VariableList`

- Do not modify the class and member function declaration.
- You can not use any C/C++ library functions and classes, other than `malloc`, `new`, `free`, `delete`, `std::string`, functions provided in “string”.
- Do not include any other header files.
- Do not use `string` class
- Do not add additional source files. All of your implementations should be included in the two files, “hw2.cpp” and “hw2.h”
- All the implementations should be executed without any error if there is no assumption, e.g., your implementation should check all corner cases that may cause segmentation faults.

## ***Where to implement?***

- You can implement your assignment in the parts specified with “IMPLEMENT HERE” comment.
- You can also delete the lines with “CHANGE HERE” to complete your implementation.
- If you want to use additional member functions or variables, make them “private”
- You can use additional static functions and forward declaration; however, it is NOT ALLOWED to use global/static variables.

## Due

---

Submit your implementation before Oct 23, Friday, 11:59:59pm, to LMS.  
We DO NOT allow late submission.

## Submission

---

Submit the source(.cpp) and header (.h) files that have your implementation.

- Compress two files, “hw2.cpp” and “hw2.h”, in a single zip file, naming “hw2\_YOURSTUDENTID.zip”
- DO NOT submit the “main.cpp” file

# Class Specification

---

## *What does the class do?*

Implement a class, called “VariableList”, which performs simple list functionalities.

- We can add new values to the list as new elements, remove an element, and replace an element with a new value.
- It should also support appending another list at the end of the list.
- ~~- A key difference from the typical lists, e.g., python list and c++ vector, is that -- Our VariableList elements of our list can have variable mixed types -- integer, floating-point value, and std::string class.~~
- A key difference from the typical lists, e.g., c++ vector, is that our VariableList elements can have variable mixed types like python list -- integer, floating-point value, and std::string class.
- For example, a single VariableList can have [1, 1.3f, “Apple”, 2, 3]

## *Member functions to implement*

1. Constructor and Destructor

```
// Constructors
VariableList();
VariableList(const int* initialArray, const int size);
VariableList(const float* initialArray, const int size);
VariableList(const std::string* initialArray, const int size);

// Destructor
~VariableList();
```

- a. You should initialize your member variables in constructors.
- b. There are three constructors having parameters that set initial elements of VariableList. Note that you should **allocate new space and copy** each array element of the given initialArray.
  - i. Otherwise, we cannot hold the values in our list when the values at the given array address are changed later.
- c. The destructor should delete all allocated memory.

2. Adding a new element at the end of the list

```
void add(const int val);  
void add(const float val);  
void add(const std::string& val);
```

3. Appending all elements of `varList` at the end of the current `VariableList` instance

```
void append(const VariableList& varList);
```

- a. You should also allocate new memory space and copy the elements to hold the values
- b. Assume the self class instance is not given with `varList`, (i.e., `&varList!=self`)

4. Replacing the value at the given index of the current `VariableList` instance

```
bool replace(const int idx, const int val);  
bool replace(const int idx, const float val);  
bool replace(const int idx, const std::string& val);
```

- a. The three functions replace the value of the given index
- b. Note that the type of the element in the current list instance would be different from the type of `val`, requiring to delete the memory space that turns to be unused.
- c. Return `false` if out-of-index; `true` otherwise.

5. Removing an element from the current instance

```
bool remove(const int idx);
```

- a. Remove the element at the given index, i.e., `idx`
- b. You also want to delete the memory space that turns to be unused.
- c. Return `false` if out-of-index; `true` otherwise.

6. Returning the number of elements in the list

```
unsigned int getSize() const;
```

7. Returning the type of an element for the given index

```
DataType getType(const int idx) const;
```

- a. `DataType` is defined in the skeleton as follows:

```
enum class DataType { Integer, Float, Str, NotAvailable };
```

- b. Return `NotAvailable` if out-of-index

8. Copy the value of an element for the given index

```
bool getValue(const int idx, int& val) const;  
bool getValue(const int idx, float& val) const;  
bool getValue(const int idx, std::string& val) const;
```

- a. The three functions copy the element value at the given index to `val`  
b. Return `false` if the type of the element value is different from the type of `val` or out-of-index; `true` otherwise, i.e., when successfully copying the value

***Some extra comments***

- I recommend thinking deeply to come up with the idea of the implementation details before diving into the actual coding
  - For example, you should determine how many arrays are required as new member variables, how to reuse some member functions in implementing other functions, etc
- You do not need to implement class copy operators. The skeleton already deletes the default copy operators to prevent mistakes:

```
VariableList(const VariableList&) = delete;  
VariableList& operator=(const VariableList&) = delete;
```

# Sample

---

## Notes

- We will check with other example inputs for grading.
- We provide the `printList` function for testing purposes.
- Sample 1~4 and similar-level examples will have 1~2 point out of 25.
- Sample 5 and similar-level examples will have 3~6 points out of 25.

## Sample Code 1

```
VariableList varList;  
varList.add(1);  
varList.add(3);  
varList.add(5);  
varList.add(7);  
printList(varList);  
  
varList.remove(0);  
printList(varList);
```

## Outputs of Sample code 1

```
1, 3, 5, 7  
1, 5, 7
```

## Sample Code 2

```
VariableList varList;  
varList.add(1.0f);  
varList.add("Carrot");  
varList.add(3);  
varList.add(4);  
varList.remove(0);  
varList.remove(0);  
varList.remove(0);  
printList(varList);
```

## Outputs of Sample code 2

```
4
```

### Sample Code 3

```
VariableList varList;  
VariableList varList2;  
varList.add(1);  
varList.add(2);  
varList.add(3);  
varList2.add(4);  
varList2.add(5.1f);  
varList.append(varList2);  
printList(varList);
```

### Outputs of Sample code 3

```
1, 2, 3, 4, 5.1
```

### Sample Code 4

```
VariableList varList;  
varList.add(1);  
varList.add(2.1f);  
varList.add(3);  
varList.replace(2, "Apple");  
printList(varList);
```

### Outputs of Sample code 4

```
1, 2.1, "Apple"
```

## Sample Code 5

```
int initialArray[] = { 1, 2, 3, 4 };
VariableList varList(initialArray, sizeof(initialArray) / sizeof(int));
printList(varList);

std::cout << "Add 1" << std::endl;
varList.add(1);
printList(varList);

std::cout << "Add 2.1f" << std::endl;
varList.add(2.1f);
printList(varList);

std::cout << "Appended another list with strings" << std::endl;
std::string anotherArray[] = { "Apple", "Banana" };
VariableList anotherVarList(anotherArray, 2);
varList.append(anotherVarList);
printList(varList);

std::string anotherStr = "Cucumber";
std::cout << "Replace Index=1 element with \"Cucumber\"" << std::endl;
varList.replace(1, anotherStr);
printList(varList);

std::cout << "Replace Index=7 element with 4.5" << std::endl;
varList.replace(7, 4.5f);
printList(varList);

std::cout << "Remove an element of Index=6" << std::endl;
varList.remove(6);
printList(varList);

std::cout << "Remove elements of Index=6 and Index=0 again" << std::endl;
varList.remove(6);
varList.remove(0);
printList(varList);

std::cout << "Try to replace an element of out-of-index" << std::endl;
bool ret = varList.replace(5, 1);
std::cout << "Return value: " << std::boolalpha << ret << std::endl;

std::cout << "Get the list size" << std::endl;
std::cout << varList.getSize() << std::endl;

std::cout << "Change a string already added and add the changed one" << std::endl;
anotherStr = "Carrot";
varList.add(anotherStr);
printList(varList);
```

## Outputs of Sample code 5

```
1, 2, 3, 4
Add 1
1, 2, 3, 4, 1
Add 2.1f
1, 2, 3, 4, 1, 2.1
Appended another list with strings
1, 2, 3, 4, 1, 2.1, "Apple", "Banana"
Replace Index=1 element with "Cucumber"
1, "Cucumber", 3, 4, 1, 2.1, "Apple", "Banana"
Replace Index=7 element with 4.5
1, "Cucumber", 3, 4, 1, 2.1, "Apple", 4.5
Remove an element of Index=6
1, "Cucumber", 3, 4, 1, 2.1, 4.5
Remove elements of Index=6 and Index=0 again
"Cucumber", 3, 4, 1, 2.1
Try to replace an element of out-of-index
Return value: false
Get the list size
5
Change a string already added and add the changed one
"Cucumber", 3, 4, 1, 2.1, "Carrot"
```