Creative Software Design, Assignment 8-2

Deadline: 2024-10-29 23:59 (No score for late submission)

- Submit your homework by uploading your zip file to the LMS assignment section. Below is an example.

```
13178_Assignment1-1_2024123456.zip
├─ 1.cc
├─ 2.cc
├─ 3.cc
└─ ...
```

- Your zip file name should follow this format: **13178_Assignment[Assignment-number]_[Student-ID].zip**
  - Ex. 13178_Assignment1-1_2024123456.zip

- Source files should be named as **<filename>.cc** *or* **<filename>.cpp**

- **You must submit your solution in the zip file before the deadline.**

1. **Implement a C++ Program for a Custom Vector Data Structure.**

A. Task Overview:

1. Implement the `MyVector` class by inheriting from the provided `Container` class.
2. Avoid using STL containers or any external libraries for the `MyVector` implementation.

B. You are given the following `Container` class definition, which should not be modified.

```
class Container {
public:
    virtual void push_back(int value) = 0;
    virtual void pop_back() = 0;
    virtual int front() = 0;
    virtual int back() = 0;
    virtual int getVectorArr(int index) = 0;
    virtual void setVectorArr(int index, int value) = 0;
};
```

C. `MyVector` Class Requirements:

1. **Data Handling:**
   i. Use a **dynamic array** to store integers.
   ii. The array size should be set during object construction.
2. **Member Functions:**
   i. Implement the following functions as per the `Container` interface:
      - `push_back()`, `pop_back()`, `front()`, `back()`, `getVectorArr()`, `setVectorArr()`.

   ii. Double the array's capacity when the array is full during `push_back`().
   iii. Print "`error`" and handle underflow when `pop_back()` is called on an empty vector.
3. **Error Handling:**
   i. For `getVectorArr()` and `setVectorArr()`, ensure that the index is in the range 0 <= `index` < `current_size`. If the index is out of bounds, print "`error`" and exit the program (use `exit(EXIT_FAILURE)`).

4. **Function Behavior:**

    i. `front()`: Returns the first element.

    ii. `back()`: Returns the last element.

    iii. `getVectorArr(int index)`: Returns the element at the specified index.

    iv. `setVectorArr(int index, int value)`: Sets the value at the specified index.

D. **Example of the `main()` function.**

```
int main() {
    Container* v = new MyVector(5);

    for (int i = 0; i < 10; i++) {
        v->push_back(i * 10);
    }

    std::cout << v->front() << std::endl;
    std::cout << v->back() << std::endl;
    std::cout << v->getVectorArr(3) << std::endl;

    v->setVectorArr(3, 20);
    std::cout << v->getVectorArr(3) << std::endl;

    for (int j = 0; j < 10; j++) {
        v->pop_back();
    }

    delete v;
    return 0;
}
```

E. Example output of your program (Bold text indicates user input):

```
0
90
30
20
90 80 70 60 50 40 20 20 10 0
```

F. Submission file: one C++ source file (File name: 1.**cc** or 1.**cpp**)

2. Write a C++ Program to Implement Pizza and Chicken Classes.

A. Implement `Pizza` and `Chicken` classes by inheriting the given `DeliveryFood` class.

1. The data type of `pizzaSideMenuList` and `chickenSideMenuList` should be `map<std::string, int>`, and the data should store the side menu name and price as private members of **`Pizza`** and **`Chicken`**.
2. The data type of price should be `int` and store the pizza or chicken price as a private member of **`Pizza`** and **`Chicken`**.
3. Implement the constructor, destructor, and the functions `addFood()`, `sideMenuAdd()`, and `showSideMenu()` for both classes.
4. The constructor of **Pizza** should set the price and initialize the side menu as follows:
   i. `{cola, 2000},{fried potato, 3000},{sprite, 2000}.`
5. The constructor of **Chicken** should set the price and initialize the side menu as follows:
   i. `{cheese ball, 4000},{cola, 2000},{fried potato, 3000}, {sprite, 2000}.`

B. You are given the following `DeliveryFood` class definition, which should not be modified:

```
class DeliveryFood {
protected:
    int totalPrice;
    std::string sideMenu;

public:
    DeliveryFood() : totalPrice(0) {}

    int getTotalPrice() { return totalPrice; }
    std::string getSideMenu() { return sideMenu; }

    virtual void sideMenuAdd(std::string _sideName) = 0;
    virtual void showSideMenu() = 0;
    virtual void addFood() = 0;
};
```

C. **Implement `main()` and `chooseSideMenu()` function.**

1. The return type of `chooseSideMenu()` function is `void`.
2. When choosing the side menu, `chooseSideMenu(DeliveryFood)` should be called.

D. **Example of the `main()` function.**

```
int main() {
   DeliveryFood* margheritaPizza = new Pizza(12000);
   DeliveryFood* gorgonzolaPizza = new Pizza(15000);
   DeliveryFood* friedChicken = new Chicken(17000);
   DeliveryFood* spicyChicken = new Chicken(19000);

   int chooseFoodNum = 0;
   std::cout << "1. margherita pizza" << std::endl;
   std::cout << "2. gorgonzola pizza" << std::endl;
   std::cout << "3. fried chicken" << std::endl;
   std::cout << "4. spicy chicken" << std::endl;
   std::cout << "choose food: ";
   std::cin >> chooseFoodNum;

   // Implement your logic to choose side menu and display
the total price
}
```

E. Example output of your program (Bold text indicates user input):

```
(Example 1)
1. margherita pizza
2. gorgonzola pizza
3. fried chicken
4. spicy chicken
choose food: 1 ⏎
1. cola 2000
2. fried potato 3000
3. sprite 2000
select side menu: 2 ⏎
margherita pizza, fried potato total price: 15000
```

```
(Example 2)
1. margherita pizza
2. gorgonzola pizza
3. fried chicken
4. spicy chicken
choose food: 3 ⏎
1. cheese ball 4000
2. cola 2000
3. fried potato 3000
4. sprite 2000
select side menu: 2 ⏎
fried chicken, cola total price: 19000
```

F. Submission file: one C++ source file (File name: 2.**cc** or 2.**cpp**)