Lab#13
Object oriented programming.
17/01/2023

Templates in C++

template is a simple yet very powerful tool in C++.

The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types.

```
#include <iostream>
using namespace std;

template <typename T> class Array {
private:
    T* ptr;
    int size;

public:
    Array(T arr[], int s);
    void print();
};

template <typename T> Array<T>::Array(T arr[], int s)
{
    ptr = new T[s];
    size = s;
    for (int i = 0; i < size; i++)
        ptr[i] = arr[i];
}

template <typename T> void Array<T>::print()
{
    for (int i = 0; i < size; i++)
        cout << " " << *(ptr + i);
    cout << endl;
}

int main()
{
    int arr[5] = { 1, 2, 3, 4, 5 };
```

```
    Array<int> a(arr, 5);
    a.print();
    return 0;
}
```

Singleton design pattern: is a software design principle that is used to restrict the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. For example, if you are using a logger, that writes logs to a file, you can use a singleton class to create such a logger. You can create a singleton class using the following code.

```
#include <iostream>

using namespace std;

class Singleton {
  static Singleton *instance;
  int data;

  // Private constructor so that no objects can be created.
  Singleton() {
    data = 0;
  }

  public:
  static Singleton *getInstance() {
    if (!instance)
    instance = new Singleton;
    return instance;
  }

  int getData() {
    return this -> data;
  }

  void setData(int data) {
    this -> data = data;
```

```
  }
};

//Initialize pointer to zero so that it can be initialized in first call to getInstance
Singleton *Singleton::instance = 0;

int main(){
  Singleton *s = s->getInstance();
  cout << s->getData() << endl;
  s->setData(100);
  cout << s->getData() << endl;
  return 0;
}
```

Factory method: is a creational design pattern, i.e., related to object creation. In the Factory pattern, we create objects without exposing the creation logic to the client and the client uses the same common interface to create a new type of object. The idea is to use a static member-function (static factory method) that creates & returns instances, hiding the details of class modules from the user. A factory pattern is one of the core design principles to create an object, allowing clients to create objects of a library (explained below) in a way such that it doesn't have a tight coupling with the class hierarchy of the library. See the following sample code.

```
#include<iostream>
using namespace std;
class Vehicle
{
        public:
                virtual void display()=0;
                virtual ~Vehicle()
                {
                }
                static Vehicle* FactoryMethod(int);
};
class Car: public Vehicle
{
        public:
                void display()
                {
                        cout<<"Car\n";
```

```
		}
};
class Bike: public Vehicle
{
	public:
		void display()
		{
			cout<<"Bike\n";
		}
};
Vehicle* Vehicle::FactoryMethod(int choice)
{
	Vehicle *vptr;
	if (choice==1)
	{

		Car c;
		vptr=&c;
		vptr->display();
	}
	else if (choice==2)
	{
		Bike b;
		vptr=&b;
		vptr->display();

	}
	else
	{
		cout<<"Not a valid choice";
	}
}

int main()
{
	Vehicle::FactoryMethod(1); //prints car
	Vehicle::FactoryMethod(2); //prints bike
	return 0;
}
```

Lab#13
Object oriented programming.
17/01/2023

Task#01

Set theory is used in everyday life, from bars to railway timetables.A set is a well-defined collection of objects. The objects may be numbers, alphabets etc. You are asked to create a class template named Sets with function(s) that perform union and intersection on 2 input arrays.

Task#02

Write a function template that finds the number of times each element exists in an array of elements.

Task#03

Computers don't understand words or numbers the way humans do. Modern software allows the end user to ignore this, but at the lowest levels of your computer, everything is represented by a binary electrical signal that registers in one of two states: "0" or "1". To make sense of complicated data, your computer has to encode it in binary. Create a C++ template class that converts input decimal numbers into binary.

Display number of objects created.

Lab#13
Object oriented programming.
17/01/2023

Task#04

Create a base Class *Bank* with 2 derived classes *Current_account* and *Saving_account*.

1. **Data members:**
    1. name(Char)
    2. Account number(int)
2. **Member function:**

    a. *getAccountDetails()* that takes name and account number from user.

    b. *displayDetails()* to display name and account number.

    c. *current_account_balace()* shows available balance in the current account and must be initialized with 1000.

    d. *current_account_deposit()* to deposit the amount in current balance.

    e. *current_account_withdraw()*to withdraw amount from account if and only if available balance in account is greater than 1000, withdrawal amount is smaller than balance in account

and after withdrawal at least 1000 is left in account else show insufficient balance.

f. *saving_account_display()*shows available balance in the saving account and must be initialized with 1000.

g. *saving_account_deposit()*to deposit the amount in available balance and add interest rate with each deposit. Formula for interest calculation is

```
interest = (sav_balance * 2) / 100;
```

h. *saving_account_withdraw()*to withdraw amount from account if and only if available balance in account is greater than 1000, withdrawal amount is smaller than balance in account and after withdrawal at least 1000 is left in account else show insufficient balance.

Implement factory design pattern for this question.