DSA LAB ASSIGNMENT

NAME: ARYAN NARANG

ROLL NO:244ca009

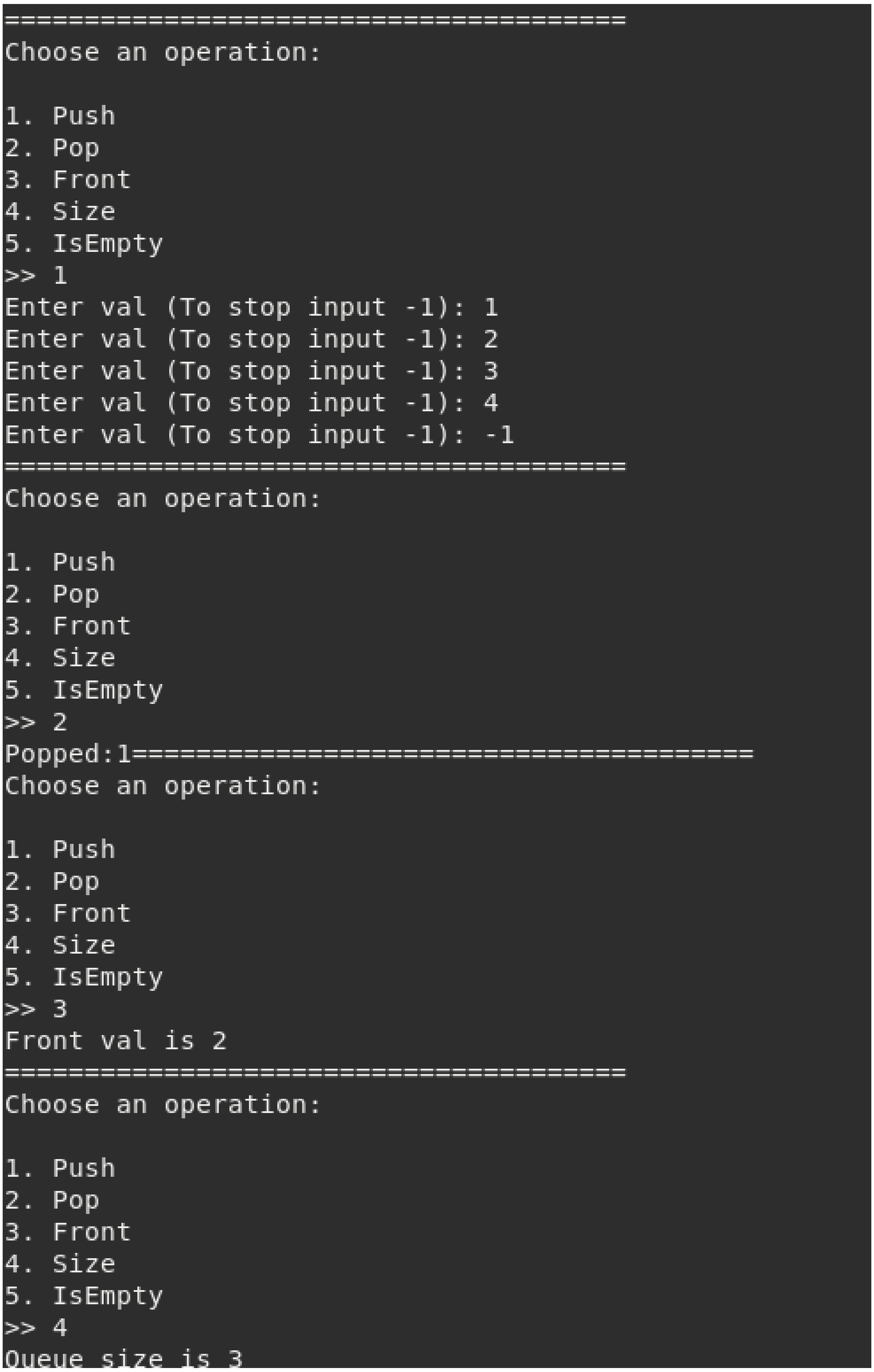
Q1. Circular queue implementation

Code:

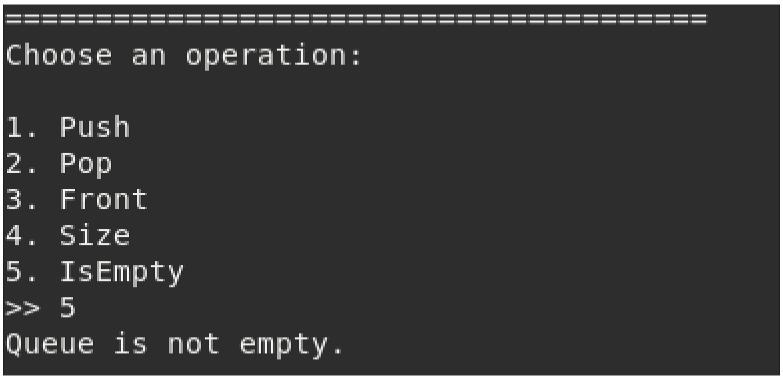
```
#include<iostream>
#define maxsize 5
using namespace std;

class Queue {
int first, last, curSize;
int arr[maxsize];
public:
      Queue() {
            first = 0;
            last = 0;
            curSize = 0;
      }
      int front() {
            if (isEmpty()) {
                  cout << "Queue is empty." << endl;
                  return -1;
            }
            return arr[first];
      }
      void pop() {
            if (isEmpty()) {
                  cout << "Queue is empty." << endl;
                  return;
            }
            cout<<"Popped:"<<front();
            first = (first+1) % maxsize;
            curSize--;
      }
      void push(int val) {
            if (curSize >= maxsize) {
                  cout << "Queue is full." << endl;
                  return;
            }
            arr[last] = val;
            last = (last+1) % maxsize;
            curSize++;
      }
      bool isEmpty() {
            return curSize == 0;
      }
      int size() {
            return curSize;
      }
};
```

```
int main() {
Queue q;

bool flag = true;

while (flag) {
      cout << "=======================================" << endl;
      cout << "Choose an operation:\n" << endl;
      cout << "1. Push" << endl;
      cout << "2. Pop" << endl;
      cout << "3. Front" << endl;
      cout << "4. Size" << endl;
      cout << "5. IsEmpty" << endl;
      cout << ">> ";
      int choice, val, size;
      bool empty;
      cin >> choice;
      switch(choice) {
            case 1:
                  while(true){
                  cout << "Enter val (To stop input -1): ";
                  cin >> val;
                  if(val==-1){
                        break;
                  }
                  q.push(val);
                  }
                  val=0;
                  break;
            case 2:
                  q.pop();
                  break;
            case 3:
                  val = q.front();
                  if (val > 0) cout << "Front val is " << val << endl;
                  break;
            case 4:
                  size = q.size();
                  cout << "Queue size is " << size << endl;
                  break;
            case 5:
                  empty = q.isEmpty();
                  if (empty) {
                        cout << "Queue is empty." << endl;
                  } else {
                        cout << "Queue is not empty." << endl;
                  }
                  break;
            default:
                  flag = false;
                  break;
      }
}
return 0;
}
```

OUTPUT:



```
=======================================
Choose an operation:

1. Push
2. Pop
3. Front
4. Size
5. IsEmpty
>> 1
Enter val (To stop input -1): 1
Enter val (To stop input -1): 2
Enter val (To stop input -1): 3
Enter val (To stop input -1): 4
Enter val (To stop input -1): -1
=======================================
Choose an operation:

1. Push
2. Pop
3. Front
4. Size
5. IsEmpty
>> 2
Popped:1=======================================
Choose an operation:

1. Push
2. Pop
3. Front
4. Size
5. IsEmpty
>> 3
Front val is 2
=======================================
Choose an operation:

1. Push
2. Pop
3. Front
4. Size
5. IsEmpty
>> 4
Queue size is 3
```



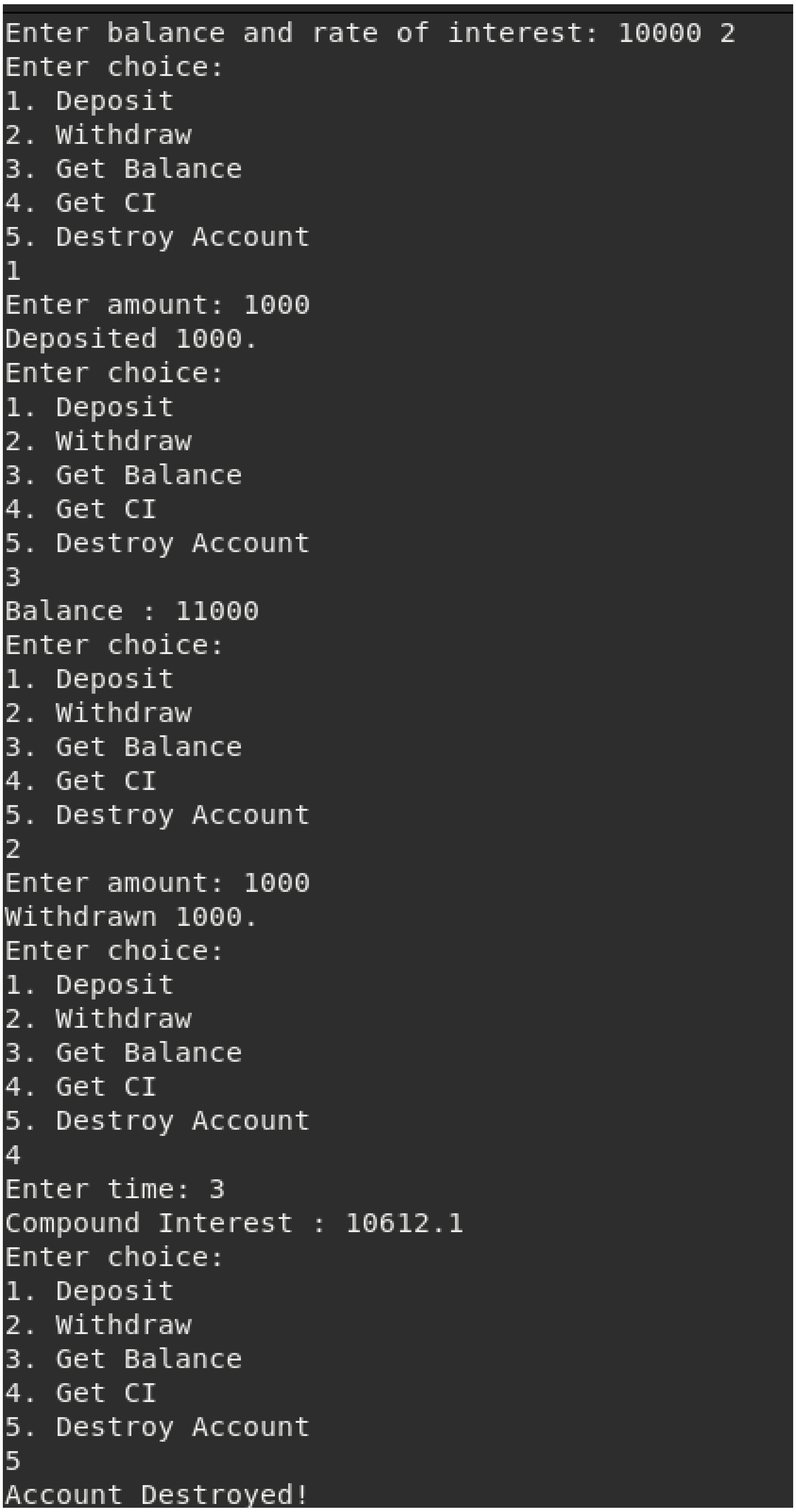
```
=======================================
Choose an operation:

1. Push
2. Pop
3. Front
4. Size
5. IsEmpty
>> 5
Queue is not empty.
```

Q2. Banking Program that came in midsem exam

Code:

```
#include<iostream>
#include<math.h>
using namespace std;

class Account {
      int balance;
      float rate;
      public:
            Account(int bal, float r) {
                  balance = bal;
                  rate = r;
            }
            void deposit(int amount) {
                  balance += amount;
                  cout << "Deposited " << amount << "." << endl;
            }
            void withdraw(int amount) {
                  if (balance < amount) {
                        cout << "Balance not enough." << endl;
                        return;
                  }
                  balance -= amount;
                  cout << "Withdrawn " << amount << "." << endl;
            }
            float calculateCI(int t) {
                  float p = balance;
                  float r = rate;
                  float ci = p * pow(1 + r/100, t);
                  return ci;
            }
            int getBalance() {
                  return balance;
            }
            void displayMenu() {
                  cout << "Enter choice:" << endl;
                  cout << "1. Deposit" << endl;
                  cout << "2. Withdraw" << endl;
                  cout << "3. Get Balance" << endl;
                  cout << "4. Get CI" << endl;
                  cout << "5. Destroy Account" << endl;
            }
            ~Account() {
                  cout << "Account Destroyed!" << endl;
            }
};

int main() {
      int balance;
      float rate;
      cout << "Enter balance and rate of interest: ";
      cin >> balance >> rate;
      Account *acc = new Account(balance, rate);
      bool flag = true;
      while (flag) {
            int choice, amount, time;
            acc->displayMenu();
            cin >> choice;
            switch(choice) {
                  case 1:
                        cout << "Enter amount: ";
                        cin >> amount;
                        acc->deposit(amount);
                        break;
                  case 2:
                        cout << "Enter amount: ";
                        cin >> amount;
                        acc->withdraw(amount);
                        break;
                  case 3:
                        cout << "Balance : " << acc->getBalance() << endl;
                        break;
                  case 4:
                        cout << "Enter time: ";
                        cin >> time;
                        cout << "Compound Interest : " << acc->calculateCI(time)
<< endl;
                        break;
                  case 5:
                        flag = false;
                        delete acc;
                        break;
                  default:
                        cout << "Invalid." << endl;
                        break;
            }
      }
      return 0;
}
```

OUTPUT:



```
Enter balance and rate of interest: 10000 2
Enter choice:
1. Deposit
2. Withdraw
3. Get Balance
4. Get CI
5. Destroy Account
1
Enter amount: 1000
Deposited 1000.
Enter choice:
1. Deposit
2. Withdraw
3. Get Balance
4. Get CI
5. Destroy Account
3
Balance : 11000
Enter choice:
1. Deposit
2. Withdraw
3. Get Balance
4. Get CI
5. Destroy Account
2
Enter amount: 1000
Withdrawn 1000.
Enter choice:
1. Deposit
2. Withdraw
3. Get Balance
4. Get CI
5. Destroy Account
4
Enter time: 3
Compound Interest : 10612.1
Enter choice:
1. Deposit
2. Withdraw
3. Get Balance
4. Get CI
5. Destroy Account
5
Account Destroyed!
```

Q3. Two single inheritance programs

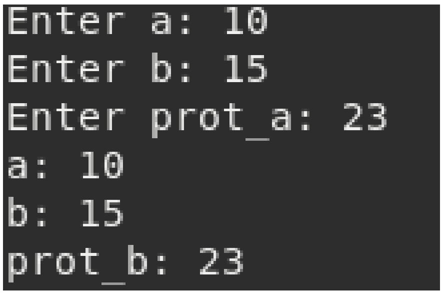
SINGLE INHERITANCE 1:

Code:

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

class A {
	int a;
	protected:
		int prot_a;
		void set_prot_a() {
			cout << "Enter prot_a: ";
			cin >> prot_a;
		}
	public:
		void set_a() {
			cout << "Enter a: ";
			cin >> a;
		}
		int get_a() {
			return a;
		}
};

class B: public A {
	int b;
	public:
		void set_b() {
			cout << "Enter b: ";
			cin >> b;
		}
		int get_b() {
			return b;
		}
		int get_prot_a() {
			return prot_a;
		}
		void set_prot() {
			set_prot_a();
		}
};

int main() {
	B b;
	b.set_a();
	b.set_b();
	b.set_prot();
```

```
      cout << "a: " << b.get_a() << endl;
      cout << "b: " << b.get_b() << endl;
      cout << "prot_b: " << b.get_prot_a() << endl;
      return 0;}
```

OUTPUT:



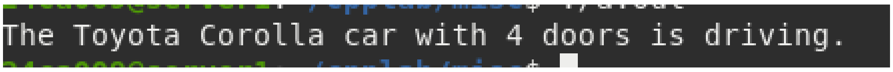
```
Enter a: 10
Enter b: 15
Enter prot_a: 23
a: 10
b: 15
prot_b: 23
```

SINGLE INHERITANCE 2:

Code:

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

class Vehicle {
public:
    string make;
    string model;

    Vehicle(string m, string mo) : make(m), model(mo) {}

    void drive() {
        cout << "The " << make << " " << model << " is driving." << endl;
    }
};

class Car : public Vehicle {
public:
    int doors;

    Car(string m, string mo, int d) : Vehicle(m, mo), doors(d) {}

    void drive() {
        cout << "The " << make << " " << model << " car with " << doors << "
doors is driving." << endl;
    }
};

int main() {
    Car myCar("Toyota", "Corolla", 4);
    myCar.drive();
    return 0;
}
```

OUTPUT:



```
The Toyota Corolla car with 4 doors is driving.
```

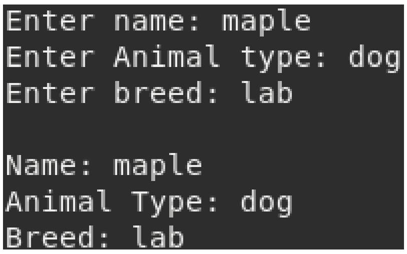
Q4. Multilevel Inheritance program

Code:

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

class Thing {
	string name;
	protected:
		void set_name() {
			cout << "Enter name: ";
			cin >> name;
		}
		void get_name() {
			cout << "Name: " << name << endl;
		}
};

class Animal: public Thing {
	string animal_type;
	protected:
		void set_animal_type() {
			cout << "Enter Animal type: ";
			cin >> animal_type;
		}
		void get_animal_type() {
			cout << "Animal Type: " << animal_type << endl;
		}
};

class Dog: public Animal {
	string breed;
	protected:
		void set_breed() {
			cout << "Enter breed: ";
			cin >> breed;
		}
		void get_breed() {
			cout << "Breed: " << breed << endl;
		}
	public:
		void set_info() {
			set_name();
```

```
                set_animal_type();
                set_breed();
            }
            void get_info() {
                get_name();
                get_animal_type();
                get_breed();
            }
};

int main() {
      Dog myDog;
      myDog.set_info();
      cout << endl;
      myDog.get_info();
}
```

OUTPUT:



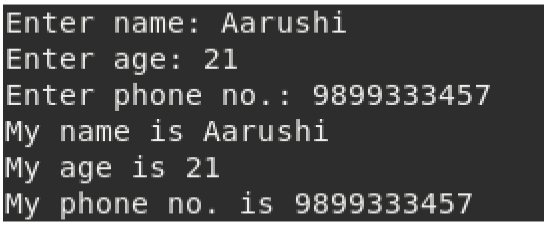
```
Enter name: maple
Enter Animal type: dog
Enter breed: lab

Name: maple
Animal Type: dog
Breed: lab
```

Q5. Hybrid Inheritance program

Code:

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

class GrandParent {
      private:
            string name;
      protected:
            string getName() {
                  return name;
            }
            void showName() {
                  cout << "My name is " << name << endl;
            }
            void setName() {
                  cout << "Enter name: ";
                  cin >> name;
            }
};

class Parent1 : virtual public GrandParent {
      private:
            int age;
      protected:
            void showAge() {
                  cout << "My age is " << age << endl;
            }
            int getAge() {
                  return age;
            }
            void setAge() {
                  cout << "Enter age: ";
                  cin >> age;
            }
};

class Parent2 : virtual public GrandParent {
      private:
            long long phoneNo;
      protected:
            void setPhoneNo() {
                  cout << "Enter phone no.: ";
                  cin >> phoneNo;
            }
            void showPhoneNo() {
                  cout << "My phone no. is " << phoneNo << endl;
            }
            long long getPhoneNo() {
                  return phoneNo;
            }
};

class Child : public Parent1, public Parent2 {
      private:
            int gender;
      public:
            void setInfo() {
                  setName();
                  setAge();
                  setPhoneNo();
            }
            void getInfo() {
                  showName();
                  showAge();
                  showPhoneNo();
            }
};

int main() {
      Child c;
      c.setInfo();
      c.getInfo();
      return 0;
}
```

OUTPUT:



```
Enter name: Aarushi
Enter age: 21
Enter phone no.: 9899333457
My name is Aarushi
My age is 21
My phone no. is 9899333457
```