Bank Account Management.

॥वसुधैव कुटुम्बकम्॥
SYMBIOSIS
INSTITUTE OF TECHNOLOGY, PUNE

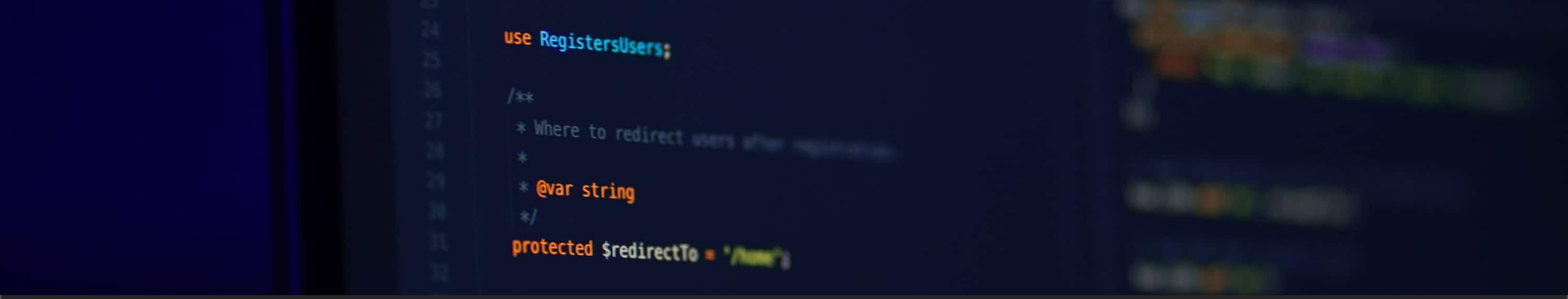
Computer Science Engineering Department
Symbiosis Institute of Technology, Pune

Name-Shubham Upadhyay

PRN-22070122209

What is C++?

C++ is a powerful programming language widely used in software development. It is an extension of the C programming language and adds object-oriented programming (OOP) features to it. C++ is known for its efficiency, versatility, and performance. It is used to develop a wide range of applications, including operating systems, video games, and desktop applications. In addition to OOP, C++ supports other programming paradigms such as procedural programming and generic programming.



Benefits of Using C++

1 Performance

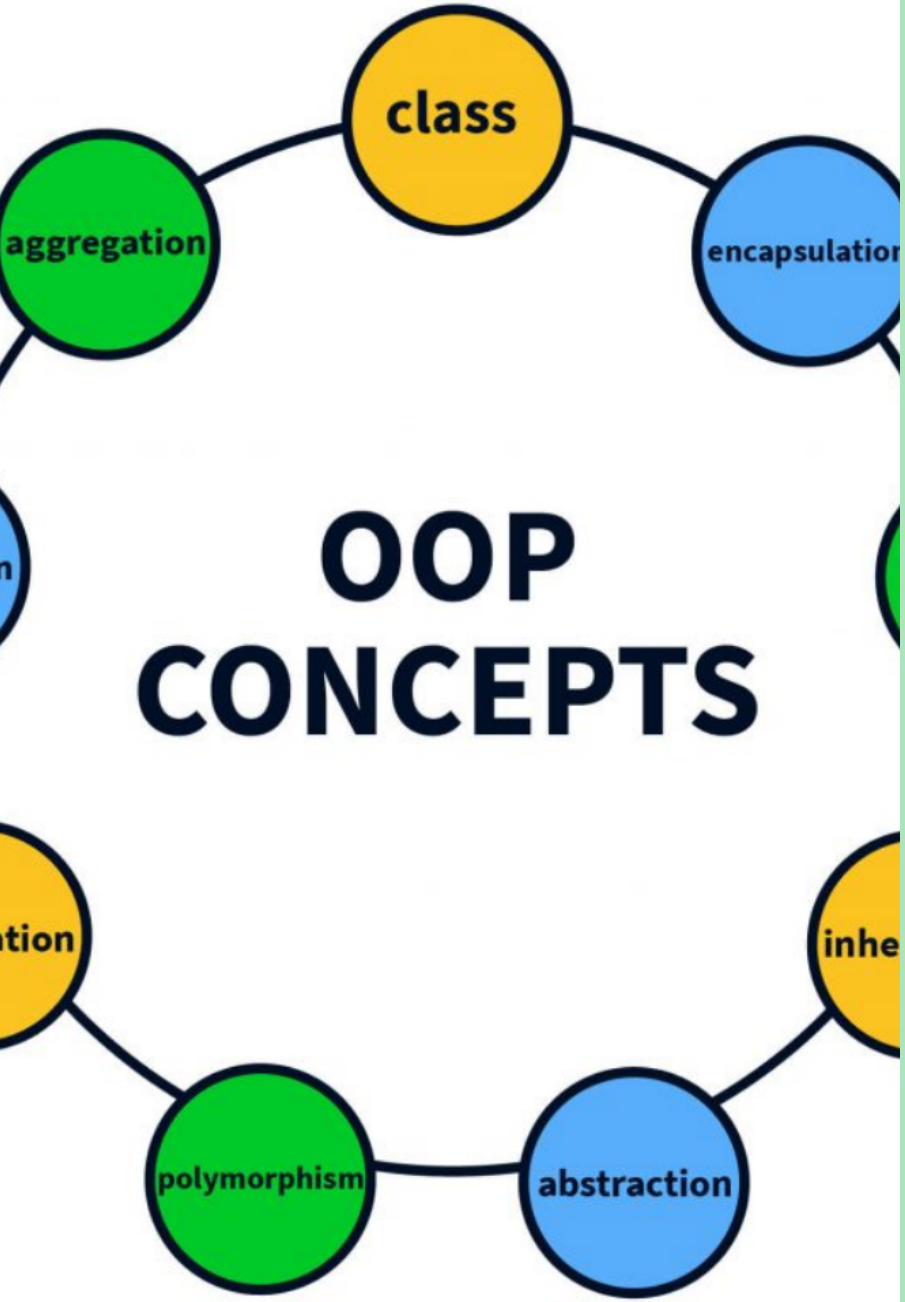
C++ offers high performance and efficiency, ensuring quick and reliable account management.

2 Flexibility

With C++, you can customize bank account management systems to meet unique requirements.

3 Security

Utilizing C++ provides robust security measures, safeguarding sensitive financial data.



Introduction to OOPS

Object-Oriented Programming (OOPS) is a programming paradigm that organizes data and behavior into reusable structures called objects. It focuses on creating objects that have properties (attributes) and behaviors (methods) and allows these objects to interact with each other to build complex applications. OOPS provides concepts such as encapsulation, inheritance, and polymorphism, which enhance code reusability, maintainability, and modularity.

Understanding OOPS Concepts

Learn the fundamental concepts of Object-Oriented Programming (OOPS) and how they can be applied to enhance software development. Dive into encapsulation, abstraction, inheritance, and polymorphism to build robust and flexible applications.

Encapsulation

Encapsulation is the process of bundling data and methods together in a class, hiding the internal details and providing a public interface to interact with the object. It helps in achieving data abstraction, data hiding, and modularity in code.

Abstraction

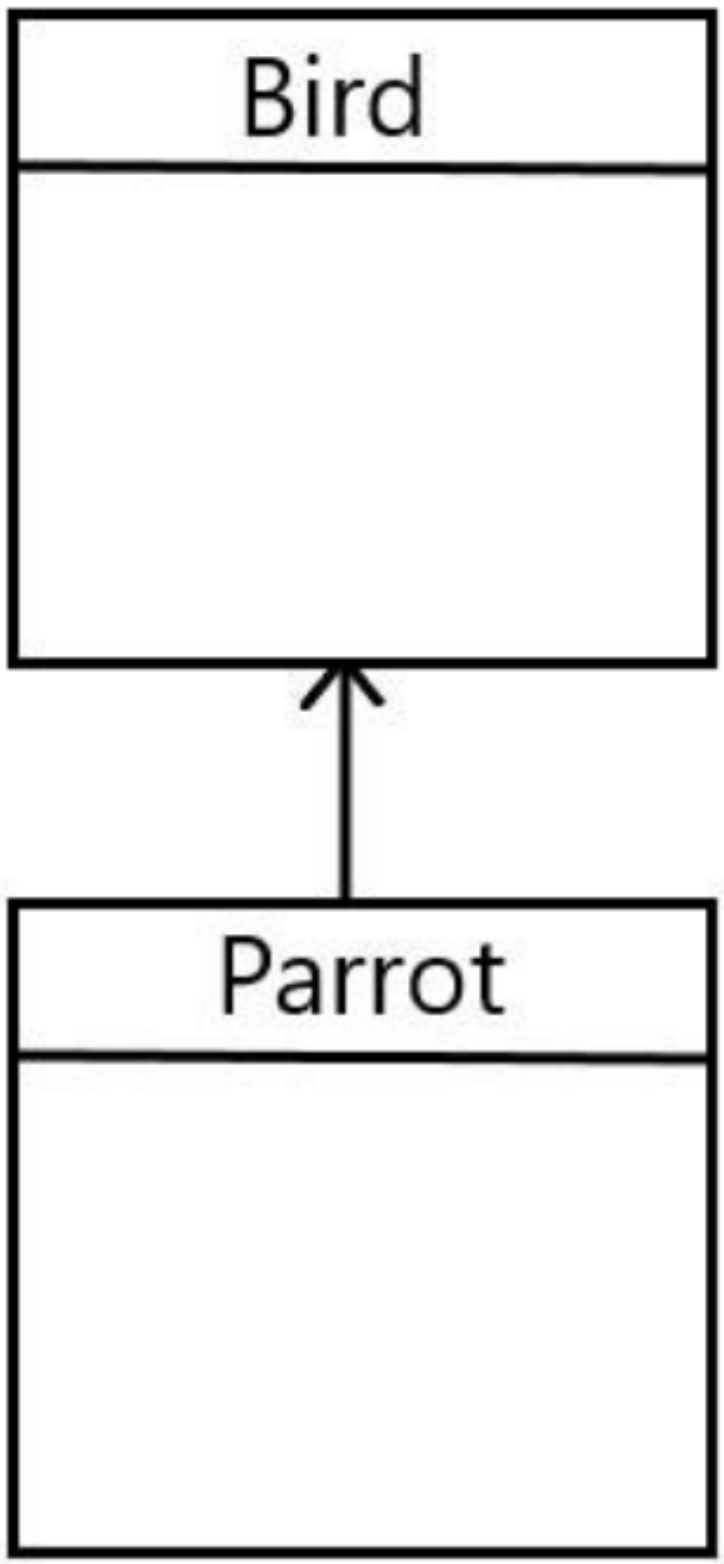
Abstraction focuses on representing the essential features of an object, while hiding the unnecessary details. It allows programmers to create classes and objects that model real-world entities, making code more maintainable and reusable.

Inheritance

Inheritance is a mechanism in object-oriented programming that allows one class to inherit properties and methods from another class. It promotes code reusability, enables the creation of hierarchies, and facilitates the implementation of the "is-a" relationship between classes.

Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the same method to be used for different types of objects, providing flexibility and extensibility in code design. Polymorphism plays a crucial role in achieving code reusability, enhancing the scalability of applications, and enabling dynamic method dispatch.



Inheritance in Object-Oriented Programming

Inheritance allows you to create new classes based on existing classes, inheriting their properties and behaviors. It promotes code reuse and enhances code organization and maintainability.

For example, consider a class hierarchy for animals. The base class could be "Animal", and derived classes like "Dog", "Cat", and "Bird" inherit the common characteristics and behaviors of an animal while adding their own unique features.

Example of inheritance:

```
class Base {
  public:
    float salary = 900;
};
class Derived: public Base {
  public:
    float bonus = 100;
    void sum() {
      cout << "Your Total Salary is: " << (salary + bonus) << endl;
    }
};
int main() {

  // Creating an object of the derived class.
  Derived x;

  // Gets the salary variable of Base class.
  cout << "Your Salary is:" << x.salary << endl;
  // Gets the bonus variable of the Derived class.
  cout << "Your Bonus is:" << x.bonus << endl;
  x.sum();
  return 0;
}
```

Role of Inheritance

Code Reusability

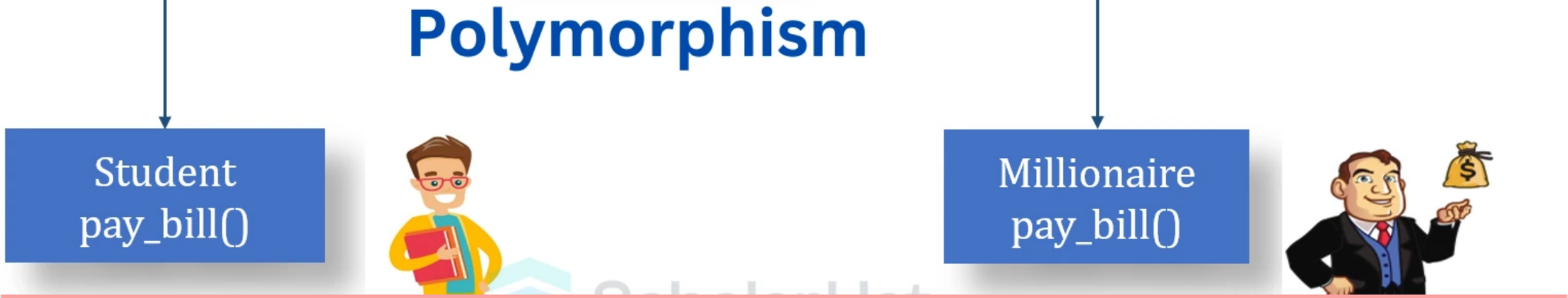
Using inheritance in bank account management allows for efficient code reuse, reducing redundancy.

Enhanced Organization

Inheritance provides a structured approach to organizing different types of bank accounts.

Scalability

Inheritance facilitates scalability by allowing the creation of new account types effortlessly.



Understanding Polymorphism

1

Polymorphic Objects

A polymorphic object is **an object that is capable of taking on multiple forms**. The kind of polymorphism the object undergoes depends on when the object takes its form and what part of the object is transforming.

2

Dynamic Binding

Run time polymorphism is also known as Dynamic Method Dispatch as **the method functionality is decided dynamically at run time based on the object**. It is also called “Late Binding” as the process of binding the method with the object occurs late after compilation.

3

Code Maintenance

Polymorphism simplifies code maintenance by reducing the need for multiple conditional statements.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 10;
    int b = 32;

    cout << "Value of a + b is: " << a + b;
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    string a = "poly";
    string b = "morphism";

    cout << "Value of a + b is: " << a + b;
```

in the first code + is used to add a=10 and *b*=32, the output Value of *a+b* is : 42.

in the second code + is used to concatenate two strings *a* = poly and *b* = morphism, therefore the output Value of *a+b* is polymorphism.

Implementing Bank Account Management

Bank Account Management system uses different C++ programming concepts and OOPS concepts, the system is created using concepts of class and objects ,inheritance and polymorphism have used a person class and inherited a class bankaccount for accounts.

The system has many functionalities such as create account,deposit money,withdraw money, change password.We can also view account holder details.

CODE: Bank Account Management

```cpp
#include <iostream>
#include <string>
#include <cctype>

using namespace std;

class Person {
public:
    string name;
    long ID;
    string address;
    string contact;

public:
    Person() {}

    Person(const string& n, long id, const string& addr, const string& con)
        : address(addr), contact(con) {
        if (validateName(n)) {
            name = n;
        } else {
            cout << "Invalid name format. Name must be a valid string." << endl;
            name = "Invalid Name";
        }
        ID = id;
        if (!validateContact(contact)) {
            cout << "Invalid contact number. Contact must be a 10-digit number." << endl;
            contact = "Invalid Contact";
        }
    }

    bool validateName(const string& str) {
        for (char c : str) {
            if (!isalpha(c) && !isspace(c)) {
                return false;
            }
        }
        return true;
    }

    bool validateContact(const string& contact) {
        if (contact.length() != 10) {
            return false;
        }
        for (char c : contact) {
            if (!isdigit(c)) {
                return false;
            }
        }
        return true;
    }

    void display() {
        cout << "Name: " << name << endl;
        cout << "ID: " << ID << endl;
        cout << "Address: " << address << endl;
        cout << "Contact: " << contact << endl;
    }
};

class BankAccount : public Person {
private:
    long cash;
    string password;

public:
    BankAccount() : cash(0) {}

    BankAccount(const string& n, long id, const string& addr, const string& con, long money, const string& pass)
        : Person(n, id, addr, con), cash(money), password(pass) {}

    void display() {
        Person::display();
        cout << "Cash: " << cash << endl;
    }

    void deposit(long amount) {
        cash += amount;
        cout << "Deposited " << amount << " into the account. New balance: " << cash << endl;
    }

    bool validatePassword(const string& inputPassword) {
        return password == inputPassword;
    }

    void changePassword(const string& oldPassword, const string& newPassword) {
        if (validatePassword(oldPassword)) {
            password = newPassword;
            cout << "Password changed successfully." << endl;
        } else {
            cout << "Incorrect previous password. Password not changed." << endl;
        }
    }

    void withdraw(long amount) {
        cout << "Enter your password for verification: ";
        string inputPassword;
        cin >> inputPassword;

        if (validatePassword(inputPassword)) {
            if (amount <= cash) {
                cash -= amount;
                cout << "Withdrawn " << amount << " from the account. New balance: " << cash << endl;
            } else {
                cout << "Insufficient funds for withdrawal." << endl;
            }
        } else {
            cout << "Incorrect password. Withdrawal not allowed." << endl;
        }
    }
};

int main() {
    BankAccount accounts[100];
    int total = 0;

    while (true) {
        cout << "Choose an option:" << endl;
        cout << "1. Create a new account" << endl;
        cout << "2. View customer details" << endl;
        cout << "3. Deposit to account" << endl;
        cout << "4. Withdraw from account" << endl;
        cout << "5. Change password" << endl;
        cout << "6. Exit" << endl;

        int choice;
        cin >> choice;

        switch (choice) {
            case 1: {
                // Create a new account
                string name, address, password, contact;
                long ID, cash;

                cout << "Enter name: ";
                cin.ignore();
                getline(cin, name);
```

```
            cin.ignore();
            getline(cin, name);

            bool containsDigits = false;
            for (char c : name) {
                if (isdigit(c)) {
                    containsDigits = true;
                    break;
                }
            }

            if (containsDigits) {
                cout << "Invalid input for name. Name must be a valid string without digits." << endl;
                break;
            }

            cout << "Enter ID: ";
            cin >> ID;
            cout << "Enter address: ";
            cin.ignore();
            getline(cin, address);
            cout << "Enter contact (10 digits): ";
            cin >> contact;

            if (!accounts[0].validateContact(contact)) {
                cout << "Invalid contact number. Contact must be a 10-digit number." << endl;
                break;
            }

            cout << "Enter password: ";
            cin >> password;
            cout << "Enter initial cash balance: ";
            cin >> cash;

            BankAccount account(name, ID, address, contact, cash, password);
            accounts[total] = account;
            total++;
            cout << "Account created successfully." << endl;
            break;
        }
        case 2: {
            // View customer details
            long searchID;
            cout << "Enter the ID of the customer: ";
            cin >> searchID;

            bool found = false;
            for (int i = 0; i < total; i++) {
                if (accounts[i].ID == searchID) {
                    accounts[i].display();
                    found = true;
                    break;
                }
            }

            if (!found) {
                cout << "Customer not found." << endl;
            }
            break;
        }
        case 3: {
            // Deposit to account
            long depositID;
            long amount;
            cout << "Enter the ID of the customer to deposit to: ";
            cin >> depositID;

            bool found = false;
            for (int i = 0; i < total; i++) {
                if (accounts[i].ID == depositID) {
                    cout << "Enter the amount to deposit: ";
                    cin >> amount;
                    accounts[i].deposit(amount);
                    found = true;
                    break;
                }
            }

            if (!found) {
                cout << "Customer not found." << endl;
            }
            break;
        }
        case 4: {
            // Withdraw from account
            long withdrawID;
            long amount;
            cout << "Enter the ID of the customer to withdraw from: ";
            cin >> withdrawID;

            bool found = false;
            for (int i = 0; i < total; i++) {
                if (accounts[i].ID == withdrawID) {
                    cout << "Enter the amount to withdraw: ";
                    cin >> amount;
                    accounts[i].withdraw(amount);
                    found = true;
                    break;
                }
            }

            if (!found) {
                cout << "Customer not found." << endl;
            }
            break;
        }
        case 5: {
            // Change password
            long changePasswordID;
            cout << "Enter the ID of the customer to change the password: ";
            cin >> changePasswordID;

            bool found = false;
            for (int i = 0; i < total; i++) {
                if (accounts[i].ID == changePasswordID) {
                    string oldPassword, newPassword;
                    cout << "Enter the old password: ";
                    cin >> oldPassword;

                    if (accounts[i].validatePassword(oldPassword)) {
                        cout << "Enter the new password: ";
                        cin >> newPassword;
                        accounts[i].changePassword(oldPassword, newPassword);
                        found = true;
                    } else {
                        cout << "Incorrect previous password. Password not changed." << endl;
                        found = true;
                    }

                    break;
                }
            }

            if (!found) {
                cout << "Customer not found." << endl;
            }
            break;
        }
```

```
                    break;
                }
                case 6:
                    cout << "Exiting the program." << endl;
                    return 0;
                default:
                    cout << "Invalid choice. Please try again." << endl;
            }
        }

        return 0;
    }

```

INPUT AND OUTPUT:

```
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
1
Enter name: Samuel
Enter ID: 1
Enter address: Baner,Pune,411226
Enter contact (10 digits): 7887653421
Enter password: sam12
Enter initial cash balance: 21
Account created successfully.
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
1
Enter name: John
Enter ID: 2
Enter address: Balewadi,pune,421123
Enter contact (10 digits): 9889766754
Enter password: john23
Enter initial cash balance: 34
Account created successfully.
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
2
Enter the ID of the customer: 1
Name: Samuel
ID: 1
Address: Baner,Pune,411226
Contact: 7887653421
Cash: 21
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
3
Enter the ID of the customer to deposit to: 2
Enter the amount to deposit: 34
Deposited 34 into the account. New balance: 68
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
4
Enter the ID of the customer to withdraw from: 1
Enter the amount to withdraw: 4
Enter your password for verification: sam12
Withdrawn 4 from the account. New balance: 17
Choose an option:
1. Create a new account
```

```
Withdrawn 4 from the account. New balance: 17
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
5
Enter the ID of the customer to change the password: 1
Enter the old password: sam2
Incorrect previous password. Password not changed.
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
5
Enter the ID of the customer to change the password: 1
Enter the old password: sam12
Enter the new password: sammy12
Password changed successfully.
Choose an option:
1. Create a new account
2. View customer details
3. Deposit to account
4. Withdraw from account
5. Change password
6. Exit
6
Exiting the program.
```

Conclusion and Key Takeaways

1 What did we cover?

We covered a lot of concepts in this project :

- Introduction to C++
- Benefits of using C++
- Understanding OOPS concepts
- Inheritance in OOP
- Polymorphism in OOP
- Making Bank Account Management System using these concepts

2 What can we do now?

We can use concepts of OOPs and data structures and algorithms and create more advanced form of account management system we can add nominee accounts, previous loans details and credit scores and deposit and withdraw history