Department of Computer Science & Engineering

SLIET Longowal

LAB MANUAL

of

Object Oriented Programming (CS-221/PCCS-208)

for

Integrated Certificate & Diploma in Computer Science (ICD-CS) 4th Sem

Submitted by

Er. Rahul Gautam (A.P., CSE)

LIST OF EXPERIMENTS

1. Write a C++ program to print out all Armstrong numbers between 1 and 500. If sum of cubes of each digit of the number is equal to the number itself, then the number is called an Armstrong number. For example, 153 = (1*1*1) + (5*5*5) + (3*3*3).
2. Write a recursive function to obtain the running sum of first 25 natural numbers.
3. Write a menu driven C++ program which has the following options:
   1. Factorial of a number
   2. Prime or not
   3. Odd or even
   4. Exit

   Once a menu item is selected the appropriate action should be taken and once this action is finished, the menu should reappear. Unless the user selects the 'Exit' option the program should continue to run.

   Hint: make use of an infinite *while* and a *switch* statement.
4. Write a function that receives 5 integers and returns the sum, average and standard deviation of these numbers. Call this function from the *main()* and print the results in *main().*
5. Write a recursive function to obtain the first 25 numbers of a Fibonacci sequence. In a Fibonacci sequence the sum of two successive terms gives the third term. Following are the first few terms of the Fibonacci sequence:
   1 1 2 3 5 8 13 21 34 55 89…
6. Define a class to represent a bank account. Include the following members:
   Data members
   a. Name of the depositor
   b. Account Number
   c. Type of account
   d. The balance amount in the account

   Member functions
   a. To assign initial values
   b. To deposit an amount
   c. To withdraw an amount after checking the balance
   d. To display name and balance

   Write a main program to test the program.
7. Write a single C++ program that demonstrates all forms of inheritance.
8. Write a C++ program that read a file and count how many characters, spaces, tabs and newlines are present in it.
9. Ramanujan number id the smallest number that can be expressed as sum of two cubes in two different ways. Write a C++ program to print all such numbers up to a reasonable limit.

What is the Structure of a C++ Program?

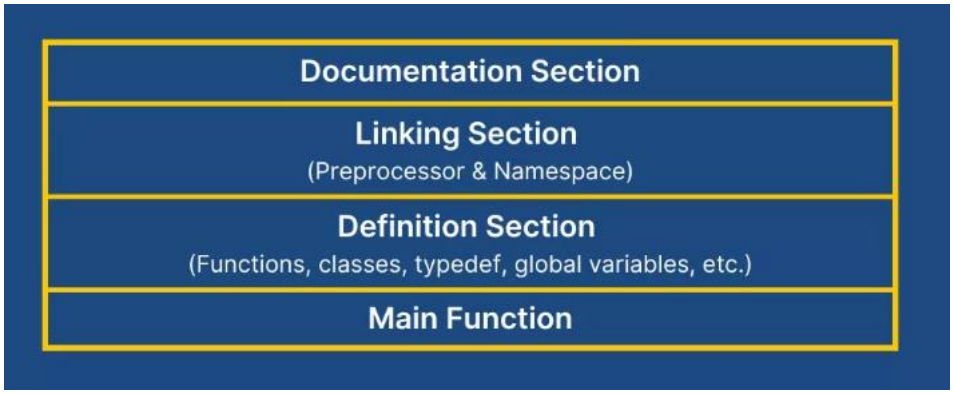

Figure: Structure of a C/C++ Program

- **Document Section:** Details about the program.
- **Linking Section:** Preprocessor directives and used namespace (standard or user defined)
- **Definition Section:** Includes all function definitions, class definitions, loops, global variables etc.
- **Main Function:** Call other functions with or without objects to accomplish the task.

Compilation of a C++ Program:

The compilation is the process of converting the source code of the C/C++ language into machine code. As C/C++ is a mid-level language, it needs a compiler to convert it into an executable code so that the program can be run on our machine.
The C program goes through the following phases during compilation:

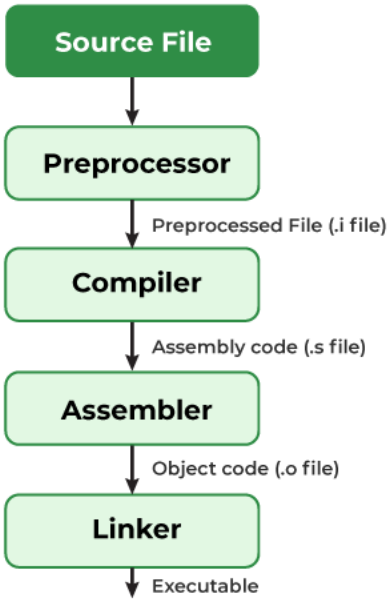

Figure: Compilation Process in C/C++ language

What are the Different IDEs available to Compile & run a C++ Program?

- Code::Blocks
- Visual Studio (VS code)
- CLion
- NetBeans
- Eclipse
- CodeLite
- Dev C++
- Turbo C++ etc.

1. **Solution**

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

int main() {
   cout << "Armstrong numbers between 1 and 500 are: ";

   for (int num = 1; num <= 500; num++) {
      int sum = 0, temp = num;
      while (temp > 0) {
         int digit = temp % 10;
         sum += digit * digit * digit;
         temp /= 10;
      }
      if (sum == num) {
         cout << num << " ";
      }
   }

   cout << endl;
   return 0;
}
```

OUTPUT:
Armstrong numbers between 1 and 500 are: 1 153 370 371 407

2. **Solution**

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

// Recursive function to calculate running sum
int runningSum(int n) {
   if (n == 0)
      return 0;
   return n + runningSum(n - 1);
}

int main() {
   int limit = 25;
   int sum = runningSum(limit);
   cout << "The running sum of the first " << limit << " natural numbers is: " << sum <<
endl;
   return 0;
}
```

OUTPUT

The running sum of the first 25 natural numbers is: 325

3. Solution

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

// Function to calculate factorial
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}

// Function to check if number is prime
bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return false;
    }
    return true;
}

// Function to check if number is odd or even
string oddOrEven(int n) {
    return (n % 2 == 0) ? "Even" : "Odd";
}

int main() {
    int choice, number;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Factorial of a number\n";
        cout << "2. Prime or not\n";
        cout << "3. Odd or even\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter a number: ";
                cin >> number;
                cout << "Factorial of " << number << " is " << factorial(number) << endl;
```

```
            break;
        case 2:
            cout << "Enter a number: ";
            cin >> number;
            cout << number << (isPrime(number) ? " is a prime number." : " is not a prime
number.") << endl;
            break;
        case 3:
            cout << "Enter a number: ";
            cin >> number;
            cout << number << " is " << oddOrEven(number) << "." << endl;
            break;
        case 4:
            cout << "Exiting program." << endl;
            return 0;
        default:
            cout << "Invalid choice. Please try again." << endl;
    }
  }
}
```

OUTPUT:

Menu:
1. Factorial of a number
2. Prime or not
3. Odd or even
4. Exit
Enter your choice: 1
Enter a number: 7
Factorial of 7 is 5040

Menu:
1. Factorial of a number
2. Prime or not
3. Odd or even
4. Exit
Enter your choice: 2
Enter a number: 7
7 is a prime number.

Menu:
1. Factorial of a number
2. Prime or not
3. Odd or even
4. Exit
Enter your choice: 3

```
Enter a number: 7
7 is Odd.

Menu:
1. Factorial of a number
2. Prime or not
3. Odd or even
4. Exit
Enter your choice: 4
Exiting program.
```

4. Solution

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

// Function to calculate sum, average, and standard deviation
void calculateStats(int arr[], int size, int &sum, double &average, double &stdDev) {
    sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    average = static_cast<double>(sum) / size;

    double variance = 0;
    for (int i = 0; i < size; i++) {
        variance += pow(arr[i] - average, 2);
    }
    variance /= size;
    stdDev = sqrt(variance);
}

int main() {
    int numbers[5];
    cout << "Enter 5 integers: ";
    for (int i = 0; i < 5; i++) {
        cin >> numbers[i];
    }

    int sum;
    double average, stdDev;
    calculateStats(numbers, 5, sum, average, stdDev);

    cout << "Sum: " << sum << endl;
    cout << "Average: " << average << endl;
```

```
    cout << "Standard Deviation: " << stdDev << endl;

    return 0;
}
```

OUTPUT:

```
Enter 5 integers: 456
567
346
235
245
Sum: 1849
Average: 369.8
Standard Deviation: 126.989
```

5. Solution

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

// Recursive function to calculate Fibonacci number
int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int count = 25;
    cout << "First " << count << " numbers of the Fibonacci sequence are: ";
    for (int i = 1; i <= count; i++) {
        cout << fibonacci(i) << " ";
    }
    cout << endl;
    return 0;
}
```

OUTPUT:

First 25 numbers of the Fibonacci sequence are: 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025

6. Solution

```
#include <iostream>
#include <string>
```

```
using namespace std;

class BankAccount {
private:
   string name;
   string accountNumber;
   string accountType;
   double balance;

public:
   // Function to assign initial values
   void initialize(string accName, string accNumber, string accType, double initialBalance)
{
      name = accName;
      accountNumber = accNumber;
      accountType = accType;
      balance = initialBalance;
   }

   // Function to deposit an amount
   void deposit(double amount) {
      if (amount > 0) {
         balance += amount;
         cout << "Amount deposited successfully." << endl;
      } else {
         cout << "Invalid deposit amount." << endl;
      }
   }

   // Function to withdraw an amount after checking balance
   void withdraw(double amount) {
      if (amount > 0 && amount <= balance) {
         balance -= amount;
         cout << "Amount withdrawn successfully." << endl;
      } else {
         cout << "Invalid or insufficient balance for withdrawal." << endl;
      }
   }

   // Function to display name and balance
   void display() const {
      cout << "Depositor Name: " << name << endl;
      cout << "Balance Amount: " << balance << endl;
   }
};
```

```
int main() {
    BankAccount account;
    account.initialize("John Doe", "123456789", "Savings", 1000.0);

    account.display();

    account.deposit(500.0);
    account.display();

    account.withdraw(300.0);
    account.display();

    account.withdraw(1500.0); // Attempting to withdraw more than balance
    account.display();

    return 0;
}
```

OUTPUT:

```
Depositor Name: John Doe
Balance Amount: 1000
Amount deposited successfully.
Depositor Name: John Doe
Balance Amount: 1500
Amount withdrawn successfully.
Depositor Name: John Doe
Balance Amount: 1200
Invalid or insufficient balance for withdrawal.
Depositor Name: John Doe
Balance Amount: 1200
```

7. Solution

Here's a C++ program that demonstrates all forms of inheritance using the example of cars:

- **Single Inheritance:** `Sedan` inherits from `Car`.
- **Multilevel Inheritance:** `LuxurySedan` inherits from `Sedan`, which inherits from `Car`.
- **Multiple Inheritance:** `ElectricCar` inherits from both `Car` and `ElectricFeatures`.
- **Hierarchical Inheritance:** `SUV` and `Hatchback` both inherit from `Car`.
- **Hybrid Inheritance:** `HybridCar` combines `LuxurySedan` and `ElectricFeatures`.

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

```
// Base class
class Car {
protected:
    string brand;
public:
    void setBrand(const string &b) {
        brand = b;
    }
    void displayBrand() {
        cout << "Brand: " << brand << endl;
    }
};

// Single Inheritance
class Sedan : public Car {
protected:
    string model;
public:
    void setModel(const string &m) {
        model = m;
    }
    void displayModel() {
        cout << "Model: " << model << endl;
    }
};

// Multilevel Inheritance
class LuxurySedan : public Sedan {
private:
    bool hasSunroof;
public:
    void setSunroof(bool s) {
        hasSunroof = s;
    }
    void displayFeatures() {
        cout << "Sunroof: " << (hasSunroof ? "Yes" : "No") << endl;
    }
};

// Multiple Inheritance
class ElectricFeatures {
protected:
    int batteryLife;
public:
    void setBatteryLife(int life) {
        batteryLife = life;
```

```
    }
    void displayBattery() {
        cout << "Battery Life: " << batteryLife << " km" << endl;
    }
};

class ElectricCar : public Car, public ElectricFeatures {
public:
    void displayElectricCar() {
        displayBrand();
        displayBattery();
    }
};

// Hierarchical Inheritance
class SUV : public Car {
public:
    void displaySUV() {
        cout << brand << " SUV is ready for off-road!" << endl;
    }
};

class Hatchback : public Car {
public:
    void displayHatchback() {
        cout << brand << " Hatchback is compact and efficient!" << endl;
    }
};

// Hybrid Inheritance using multiple and multilevel
class HybridCar : public LuxurySedan, public ElectricFeatures {
public:
    void displayHybridCar() {
        displayBrand();
        displayModel();
        displayFeatures();
        displayBattery();
    }
};

int main() {
    cout << "--- Single Inheritance ---" << endl;
    Sedan sedan;
    sedan.setBrand("Toyota");
    sedan.setModel("Camry");
    sedan.displayBrand();
```

```
    sedan.displayModel();

    cout << "\n--- Multilevel Inheritance ---" << endl;
    LuxurySedan luxury;
    luxury.setBrand("BMW");
    luxury.setModel("7 Series");
    luxury.setSunroof(true);
    luxury.displayBrand();
    luxury.displayModel();
    luxury.displayFeatures();

    cout << "\n--- Multiple Inheritance ---" << endl;
    ElectricCar eCar;
    eCar.setBrand("Tesla");
    eCar.setBatteryLife(500);
    eCar.displayElectricCar();

    cout << "\n--- Hierarchical Inheritance ---" << endl;
    SUV suv;
    suv.setBrand("Jeep");
    suv.displaySUV();

    Hatchback hatch;
    hatch.setBrand("Volkswagen");
    hatch.displayHatchback();

    cout << "\n--- Hybrid Inheritance ---" << endl;
    HybridCar hybrid;
    hybrid.setBrand("Lexus");
    hybrid.setModel("LS 500h");
    hybrid.setSunroof(true);
    hybrid.setBatteryLife(600);
    hybrid.displayHybridCar();

    return 0;
}
```

OUTPUT:

```
--- Single Inheritance ---
Brand: Toyota
Model: Camry

--- Multilevel Inheritance ---
Brand: BMW
Model: 7 Series
```

Sunroof: Yes

--- Multiple Inheritance ---
Brand: Tesla
Battery Life: 500 km

--- Hierarchical Inheritance ---
Jeep SUV is ready for off-road!
Volkswagen Hatchback is compact and efficient!

--- Hybrid Inheritance ---
Brand: Lexus
Model: LS 500h
Sunroof: Yes
Battery Life: 600 km

8. Solution

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

int main() {
    ifstream file("input.txt");
    if (!file) {
        cout << "Unable to open file." << endl;
        return 1;
    }

    char ch;
    int characters = 0, spaces = 0, tabs = 0, newlines = 0;

    while (file.get(ch)) {
        characters++;
        if (ch == ' ')
            spaces++;
        else if (ch == '\t')
            tabs++;
        else if (ch == '\n')
            newlines++;
    }

    file.close();

    cout << "Characters: " << characters << endl;
    cout << "Spaces: " << spaces << endl;
```

```
    cout << "Tabs: " << tabs << endl;
    cout << "Newlines: " << newlines << endl;

    return 0;
}
```

OUTPUT:

//Depends upon the text file

9. Solution

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

int main() {
    int limit = 10000; // Define a reasonable limit for searching Ramanujan numbers
    unordered_map<int, pair<int, int>> cubeSums;

    cout << "Ramanujan numbers up to " << limit << " are: " << endl;

    for (int i = 1; i * i * i <= limit; i++) {
        for (int j = i; j * j * j <= limit; j++) {
            int sum = (i * i * i) + (j * j * j);
            if (cubeSums.find(sum) != cubeSums.end()) {
                cout << sum << " = " << cubeSums[sum].first << "^3 + " <<
cubeSums[sum].second << "^3 = " << i << "^3 + " << j << "^3" << endl;
            } else {
                cubeSums[sum] = {i, j};
            }
        }
    }

    return 0;
}
```

OUTPUT:

Ramanujan numbers up to 10000 are:
1729 = 1^3 + 12^3 = 9^3 + 10^3
4104 = 2^3 + 16^3 = 9^3 + 15^3