Practical No.7

Aim: Using Pointers in C++

a. Declare an integer pointer and modify value via pointer.
b. Use a pointer to access an array.
c. Dynamically allocate memory using new.
d. Deallocate memory using delete.

Name: Sandip T. Hatnore
Class:FY.BSc.CS(A)
Roll No: 37
Subject: Object Oriented Programming with C++
Sign:

A.Declare an integer pointer and modify value via pointer.

Source Code:

```
#include <iostream>
using namespace std;
int main() {
    int num = 1;          // Declare an integer variable
    int *ptr = &num;      // Declare an integer pointer and store the address of num
    cout << "Original value of num: " << num << endl;
    *ptr = 7;             // Modify the value of num using the pointer
    cout << "Modified value of num: " << num << endl;
    return 0;
}
```

Output:

```
Original value of num: 1
Modified value of num: 7

--------------------------------
```

B.Use a pointer to access an array.

Source Code:

```
#include <iostream>
using namespace std;
int main() {
    int arr[5] = {5, 10, 15, 20, 25};  // Declare an array
    int *ptr = arr;                    // Pointer pointing to the first element of the array
    cout << "Array elements using pointer: " << endl;
    for (int i = 0; i < 5; i++) {
        cout << *(ptr + i) << " ";  // Accessing elements using pointer
    }
    return 0;
}
```

Output:

```
Array elements using pointer:
5 10 15 20 25

----------------------------------
```

C.Dynamically allocate memory using new.

Source Code:

```
#include <iostream>
using namespace std;
int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    // Dynamically allocate memory for an integer array
    int *arr = new int[size];
    // Input values
    cout << "Enter " << size << " elements: ";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];  // arr[i] is same as *(arr + i)
    }
    // Display values
    cout << "You entered: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    // Free allocated memory
    delete[] arr;
    return 0;
}
```

Output:

```
Enter the size of the array: 4
Enter 4 elements: 33 23 54 66 34
You entered: 33 23 54 66
```

Practical No. 8

Aim: Implementing Inheritance

a. Write a C++ Program that illustrates single inheritance.
b. Write a C++ Program that illustrates multiple inheritance.
c. Write a C++ Program that illustrates multi-level inheritance.
d. Write a C++ Program that illustrates Hierarchical inheritance.

Name: Sandip T. Hatnore
Class: FY.BSc.CS(A)
Roll No: 37
Subject: Object Oriented Programming with C++
Sign:

A.Write a C++ Program that illustrates single inheritance.

Source Code:

```
#include <iostream>
using namespace std;
// Base class
class Parent {
public:
   void displayParent() {
      cout << "This is the Parent class." << endl;
   }
};
// Derived class (inherits from Parent)
class Child : public Parent {
public:
   void displayChild() {
      cout << "This is the Child class." << endl;
   }
};
int main() {
   // Create object of Child class
   Child obj;
   // Access method of Parent class through Child object
   obj.displayParent();
   // Access method of Child class
   obj.displayChild();
   return 0;
}
```

Output:

```
This is the Parent class.
This is the Child class.
```

B.Write a C++ Program that illustrates multiple inheritance.

Source Code:

```
#include <iostream>
using namespace std;
// First base class
class Student {
public:
   void displayStudent() {
      cout << "This is Student class." << endl;
   }
```

```
};

// Second base class
class Sports {
public:
    void displaySports() {
        cout << "This is Sports class." << endl;
    }
};
// Derived class inheriting from two base classes
class Result : public Student, public Sports {
public:
    void displayResult() {
        cout << "This is Result class, combining Student and Sports." << endl;
    }
};
int main() {
    // Create object of derived class
    Result obj;
    // Access methods from both base classes
    obj.displayStudent();
    obj.displaySports();
    // Access method from derived class
    obj.displayResult();
    return 0;
}
```

Output:

```
This is Student class.
This is Sports class.
This is Result class, combining Student and Sports.
```

C.Write a C++ Program that illustrates multi-level inheritance.

Source Code:

```
#include <iostream>
using namespace std;
// Base class
class GrandParent {
public:
    void displayGrandParent() {
        cout << "This is the GrandParent class." << endl;
    }
};
// Derived from GrandParent
class Parent : public GrandParent {
public:
    void displayParent() {
        cout << "This is the Parent class." << endl;
    }
};
// Derived from Parent
class Child : public Parent {
public:
    void displayChild() {
        cout << "This is the Child class." << endl;
    }
};
int main() {
    // Create object of Child class
    Child obj;
    // Access methods from all classes
    obj.displayGrandParent();  // From GrandParent
    obj.displayParent();       // From Parent
    obj.displayChild();        // From Child
    return 0;
}
```

Output:

```
This is the GrandParent class.
This is the Parent class.
This is the Child class.
```

D.Write a C++ Program that illustrates Hierarchical inheritance.

Source Code:

```
#include <iostream>
using namespace std;
// Base class
class Parent {
public:
    void displayParent() {
        cout << "This is the Parent class." << endl;
    }
};
// First derived class
class Child1 : public Parent {
public:
    void displayChild1() {
        cout << "This is Child1 class." << endl;
    }
};
// Second derived class
class Child2 : public Parent {
public:
    void displayChild2() {
        cout << "This is Child2 class." << endl;
    }
};
int main() {
    // Object of Child1
    Child1 obj1;
    cout << "Accessing through Child1 object:" << endl;
    obj1.displayParent();
    obj1.displayChild1();
    cout << endl;
    // Object of Child2
    Child2 obj2;
    cout << "Accessing through Child2 object:" << endl;
    obj2.displayParent();
    obj2.displayChild2();
    return 0;
}
```

Output:

```
Accessing through Child1 object:
This is the Parent class.
This is Child1 class.

Accessing through Child2 object:
This is the Parent class.
This is Child2 class.
```

Practical No.9

Aim: File Handling Operations

a. Write to a file using ofstream.
b. Read from a file using ifstream.
c. Create a binary file for student records (roll, name, marks).

Name: Sandip T. Hatnore
Class:FY.BSc.CS(A)
Roll No: 37
Subject: Object Oriented Programming with C++
Sign:

A.Write to a file using ofstream.

Source Code:

```
#include <iostream>
#include <fstream>  // Required for file handling
using namespace std;
int main() {
   // Create an ofstream object
   ofstream outFile;
   // Open a file in write mode
   outFile.open("example.txt");
   if (!outFile) {
      cout << "Error opening file!" << endl;
      return 1;
   }
   // Write to the file
   outFile << "Hello, this is a sample text file." << endl;
   outFile << "Writing data using ofstream in C++." << endl;
   // Close the file
   outFile.close();
   cout << "Data written to file successfully!" << endl;
   return 0;
}
```

Output:

```
Data written to file successfully!
```

B.Read from a file using ifstream.

Source Code:

```
#include <iostream>
#include <fstream>  // Required for file handling
#include <string>
using namespace std;
int main() {
   // Create an ifstream object
   ifstream inFile;
   // Open the file in read mode
   inFile.open("example.txt");
   if (!inFile) {
      cout << "Error opening file!" << endl;
      return 1;
   }
   string line;
   cout << "Reading from file:" << endl;
   // Read the file line by line
   while (getline(inFile, line)) {
      cout << line << endl;
   }
   // Close the file
   inFile.close();
   return 0;
}
```

Output:

```
Reading from file:
Hello, this is a sample text file.
Writing data using ofstream in C++.
```

C. Create a binary file for student records (roll, name, marks).

Source Code:

```
#include <iostream>
#include <fstream>
using namespace std;
class Student {
public:
   int roll;
   char name[50];
   float marks;
   void input() {
      cout << "Enter Roll Number: ";
      cin >> roll;
      cin.ignore();
      cout << "Enter Name: ";
      cin.getline(name, 50);
      cout << "Enter Marks: ";
      cin >> marks;
   }
   void display() {
      cout << "Roll: " << roll << ", Name: " << name << ", Marks: " << marks << endl;
   }
};
int main() {
   Student s;
   fstream file;
   // Open file in binary write mode
   file.open("student.dat", ios::out | ios::binary);
   if (!file) {
      cout << "Error opening file!" << endl;
      return 1;
   }
  int n;
   cout << "How many student records do you want to enter? ";
   cin >> n;
   cin.ignore();
   for (int i = 0; i < n; i++) {
      cout << "\nEnter details for student " << i + 1 << ":\n";
      s.input();
      file.write((char*)&s, sizeof(s));  // Write object in binary
   }
   file.close();
   cout << "\nStudent records saved to student.dat successfully!" << endl;
   return 0;
}
```

Output:

```
How many student records do you want to enter? 2

Enter details for student 1:
Enter Roll Number: 37
Enter Name: Sandip
Enter Marks: 82

Enter details for student 2:
Enter Roll Number: 24
Enter Name: Smith
Enter Marks: 84

Student records saved to student.dat successfully!
```

Practical No.10

Aim: Exception Handling

a. Demonstrate try, throw, and catch with divide-by-zero.
b. Use multiple catch blocks (int, char, string).
c. Validate input (e.g., negative age) and throw custom exceptions.

Name: Sandip T. Hatnore
Class:FY.BSc.CS(A)
Roll No: 37
Subject: Object Oriented Programming with C++
Sign:

A.Demonstrate try, throw, and catch with divide-by-zero.

Source Code:

```
#include <iostream>
using namespace std;
int main() {
   int num, den;
   cout << "Enter numerator: ";
   cin >> num;
   cout << "Enter denominator: ";
   cin >> den;
   try {
      if (den == 0) {
         throw "Division by zero is not allowed!";  // Throwing an exception
      }
      double result = (double)num / den;
      cout << "Result: " << result << endl;
   }
   catch (const char *msg) {
      cout << "Exception: " << msg << endl;  // Handling the exception
   }
   cout << "Program continues after exception handling." << endl;
   return 0;
}
```

Output:

```
Enter numerator: 07
Enter denominator: 0
Exception: Division by zero is not allowed!
Program continues after exception handling.
```

B.Use multiple catch blocks (int, char, string).

Source Code:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   int choice;
   cout << "Enter your choice (1: int, 2: char, 3: string): ";
   cin >> choice;
   try {
      if (choice == 1) {
         throw 100;  // Throw integer exception
      } else if (choice == 2) {
         throw 'A';  // Throw character exception
      } else if (choice == 3) {
         throw string("This is a string exception");  // Throw string exception
      } else {
         cout << "No exception thrown!" << endl;
      }
   }
```

```
    catch (int e) {
        cout << "Caught an integer exception: " << e << endl;
    }
    catch (char e) {
        cout << "Caught a character exception: " << e << endl;
    }
    catch (string e) {
        cout << "Caught a string exception: " << e << endl;
    }
    cout << "Program continues after exception handling." << endl;
    return 0;
}
```

Output:

```
Enter your choice (1: int, 2: char, 3: string): 1
Caught an integer exception: 100
Program continues after exception handling.
```

C.Validate input (e.g., negative age) and throw custom exceptions.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;
// Custom Exception Class
class InvalidAgeException {
public:
    string message;
    InvalidAgeException(string msg) {
        message = msg;
    }
};
int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    try {
        if (age < 0) {
            throw InvalidAgeException("Age cannot be negative!");
        }
        if (age > 150) {
            throw InvalidAgeException("Age cannot be more than 150!");
        }
        cout << "Your age is: " << age << endl;
    }
    catch (InvalidAgeException &e) {
        cout << "Exception: " << e.message << endl;
    }
    cout << "Program continues after exception handling." << endl;
    return 0;
}
```

Output:

```
Enter your age: 18
Your age is: 18
Program continues after exception handling.
```