Department Of Electronics and Communication Engineering

BEARYS INSTITUTE OF TECHNOLOGY, MANGALORE

C++ Basics
(BEC358C)

Lab Manual-2022 Scheme
(2023-24)
for
III semester

Faculty In charge
Prof. Nikhitha C
Assistant Professor
Dept of ECE

Syllabus:

C++ Basics		Semester	4
Course Code	**BEC358C**	CIE Marks	50
Teaching Hours/Week (L: T:P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	1	Exam Hours	03
Examination nature (SEE)	**Practical**		

Course objectives:

- Understand object-oriented programming concepts, and apply them in solving problems.
- To create, debug and run simple C++ programs.
- Introduce the concepts of functions, friend functions, inheritance, polymorphism and function overloading.
- Introduce the concepts of exception handling and multithreading.

Sl.No	Experiments
1	Write a C++ program to find largest, smallest & second largest of three numbers using inline functions MAX & Min.
2	Write a C++ program to calculate the volume of different geometric shapes like cube, cylinder and sphere using function overloading concept.
3	Define a STUDENT class with USN, Name & Marks in 3 tests of a subject. Declare an array of 10 STUDENT objects. Using appropriate functions, find the average of the two better marks for each student. Print the USN, Name & the average marks of all the students.
4	Write a C++ program to create class called MATRIX using two-dimensional array of integers, by overloading the operator == which checks the compatibility of two matrices to be added and subtracted. Perform the addition and subtraction by overloading + and – operators respectively. Display the results by overloading the operator <<. If (m1 == m2) then m3 = m1 +m2 and m4 = m1 – m2 else display error.
5	Demonstrate simple inheritance concept by creating a base class FATHER with data members: *First Name, Surname, DOB & bank Balance* and creating a derived class SON, which inherits: Surname & Bank Balance feature from base class but provides its own feature: First Name & DOB. Create & initialize F1 & S1 objects with appropriate constructors & display the FATHER & SON details.
6	Write a C++ program to define class name FATHER & SON that holds the income respectively. Calculate & display total income of a family using Friend function.

7	Write a C++ program to accept the student detail such as name & 3 different marks by get_data() method & display the name & average of marks using display() method. Define a friend function for calculating the average marks using the method mark_avg().
8	Write a C++ program to explain virtual function (Polymorphism) by creating a base class polygon which has virtual function areas two classes rectangle & triangle derived from polygon & they have area to calculate & return the area of rectangle & triangle respectively.
9	Design, develop and execute a program in C++ based on the following requirements: An EMPLOYEE class containing data members & members functions: i) Data members: employee number (an integer), Employee_ Name (a string of characters), Basic_ Salary (in integer), All_ Allowances (an integer), Net_Salary (an integer). (ii) Member functions: To read the data of an employee, to calculate Net_Salary & to print the values of all the data members. (All_Allowances= 123% of Basic, Income Tax (IT) =30% of gross salary (=basic_ Salary_All_Allowances_IT).
10	Write a C++ program with different class related through multiple inheritance & demonstrate the use of different access specified by means of members variables & members functions.
11	Write a C++ program to create three objects for a class named count object with data members such as roll_no & Name. Create a members function set_data ( ) for setting the data values & display( ) member function to display which object has invoked it using" this" pointer.
12	Write a C++ program to implement exception handling with minimum 5 exceptions classes including two built in exceptions.
	Course outcomes (Course Skill Set): At the end of the course the student will be able to: **1.** Write C++ program to solve simple and complex problems. **2.** Apply and implement major object-oriented concepts like message passing, function overloading, operator overloading and inheritance to solve real-world problems. **3.** Use major C++ features such as Templates for data type independent designs and File I/O to deal with large data set. **4.** Analyze, design and develop solutions to real-world problems applying OOP concepts of C++.

Program Outcomes (PO's):

1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/ Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

1. **Write a C++ program to find largest, smallest & second largest of three numbers using inline functions max & min.**

```
#include<iostream>

using namespace std;

int main()

{

int num1, num2, num3;

cout << "Enter three numbers: ";

cin >> num1 >> num2 >> num3;

int largest = max(max(num1, num2), num3);

int smallest = min(min(num1, num2), num3);

int secondLargest = num1+num2 +num3 - largest - smallest;

cout << "Largest number: " << largest << endl;

cout << "Smallest number: " << smallest << endl;

cout << "Second largest number: " <<secondLargest << endl;

return 0;

}

inline int max(int a, int b) {

return (a > b) ? a : b;

}

inline int min(int a, int b) {

return (a < b) ? a : b;

}
```

Output:

Enter three numbers: 2 4 6

Largest number: 6

Smallest number: 2

Second largest number: 4

2. **Write a C++ program to calculate the volume of different geometric shapes like cube, cylinder and sphere using function overloading concept.**

```
#include <iostream>
#include <cmath>
using namespace std;
const double PI = 3.141592653589793238;
double volume(int side) {
 return pow(side, 3);
}
double volume(double radius, double height) {
 return PI * pow(radius, 2) * height;
}
double volume(double radius) {
 return (4.0 / 3.0) * PI * pow(radius, 3);
}
int main() {
 double radius, height;
 int side;
```

```
cout << "Enter the side length of the cube: ";
 cin >> side;
 cout << "Volume of the cube: " << volume(side) << endl;

cout << "Enter the radius and height of the cylinder: ";
 cin >> radius >> height;
 cout << "Volume of the cylinder: " << volume(radius, height) << endl;

 cout << "Enter the radius of the sphere: ";
 cin >> radius;
 cout << "Volume of the sphere: " << volume(radius) << endl;
 return 0;
}
```

Output:

Enter the side length of the cube: 3

Volume of the cube: 27

Enter the radius and height of the cylinder: 4.5 5.5

Volume of the cylinder: 349.895

Enter the radius of the sphere: 6.5

Volume of the sphere: 1150.35

3. Define a STUDENT class with USN, Name & Marks in 3 tests of a subject. Declare an array of 10 STUDENT objects. Using appropriate functions, find the average of the two better marks for each student. Print the USN, Name & the average marks of all the students.

```
#include <iostream>
#include <string>
using namespace std;
const int NUM_TESTS = 3;
const int NUM_STUDENTS = 10;

class STUDENT {
private:
 string USN;
 string Name;
 int Marks[NUM_TESTS];
public:
 void setUSN(string usn) {
 USN = usn;
 }

 void setName(string name) {
 Name = name;
 }

 void setMarks(int marks[NUM_TESTS]) {
 for (int i = 0; i < NUM_TESTS; i++) {
 Marks[i] = marks[i];
 }
 }
 string getUSN() {
 return USN;
 }

 string getName() {
 return Name;
```

```
}

double calculateAverage() {

int highest1 = 0;
int highest2 = 0;
for (int i = 0; i < NUM_TESTS; i++) {
if (Marks[i] > highest1) {
highest2 = highest1;
highest1 = Marks[i];
} else if (Marks[i] > highest2) {
highest2 = Marks[i];
}
}

return (highest1 + highest2) / 2.0;
}
};


int main() {
STUDENT students[NUM_STUDENTS];

for (int i = 0; i < NUM_STUDENTS; i++) {
string usn, name;
int marks[NUM_TESTS];
cout << "Enter details for student " << i + 1 << ":" << endl;
cout << "USN: ";
cin >> usn;
cout << "Name: ";
cin.ignore();
getline(cin, name);
cout << "Enter marks for " << NUM_TESTS << " tests: ";
for (int j = 0; j < NUM_TESTS; j++) {
```

```
 cin >> marks[j];
 }
 students[i].setUSN(usn);
 students[i].setName(name);
 students[i].setMarks(marks);
 }

 cout << endl << "Student details and average marks:" << endl;
 for (int i = 0; i < NUM_STUDENTS; i++) {
cout << "Student " << i + 1 << ":" << endl;
 cout << "USN: " << students[i].getUSN() << endl;
 cout << "Name: " << students[i].getName() << endl;
 cout << "Average marks: " << students[i].calculateAverage() << endl;
 cout << endl;
 }
 return 0;
}
```

Output:

```
Enter details for student 1:
USN: 1
Name: a
Enter marks for 3 tests: 10 20 30
Enter details for student 2:
USN: 2
Name: b
Enter marks for 3 tests: 20 20 20
Enter details for student 3:
USN: 3
Name: c
Enter marks for 3 tests: 30 30 30
Enter details for student 4:
USN: 4
Name: d
```

Enter marks for 3 tests: 55 35 45
Enter details for student 5:
USN: 5
Name: e
Enter marks for 3 tests: 60 50 40
Enter details for student 6:
USN: 6
Name: f
Enter marks for 3 tests: 60 35 45
Enter details for student 7:
USN: 7
Name: k
Enter marks for 3 tests: 80 70 60
Enter details for student 8:
USN: 8
Name: l
Enter marks for 3 tests: 96 78 68
Enter details for student 9:
USN: 9
Name: kjko
Enter marks for 3 tests: 35 65 45
Enter details for student 10:
USN: 10
Name: hjtyu
Enter marks for 3 tests: 90 80 70
Student details and average marks:
Student 1:
USN: 1
Name: a
Average marks: 25

Student 2:
USN: 2
Name: b

Average marks: 20

Student 3:
USN: 3
Name: c
Average marks: 30

Student 4:
USN: 4
Name: d
Average marks: 50

Student 5:
USN: 5
Name: e
Average marks: 55

Student 6:
USN: 6
Name: f
Average marks: 52.5

Student 7:
USN: 7
Name: k
Average marks: 75

Student 8:
USN: 8
Name: l
Average marks: 87

Student 9:
USN: 9

Name: kjko
Average marks: 55

Student 10:
USN: 10
Name: hjtyu
Average marks: 85

4. Write a C++ program to create class called MATRIX using two-dimensional array of integers, by overloading the operator == which checks the compatibility of two matrices to be added and subtracted. Perform the addition and subtraction by overloading + and – operators respectively. Display the results by overloading the operator <<. If (m1 == m2) then m3 = m1 + m2 and m4 = m1 –m2 else display error.

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

int main()
{
   int r, c, a[100][100], b[100][100], sum[100][100], diff[100] [100], i, j;
   cout << "Enter number of rows (between 1 and 100): ";
   cin >> r;
   cout << "Enter number of columns (between 1 and 100): ";
   cin >> c;
   cout << "Enter elements of 1st matrix: " << endl;
   // Storing elements of first matrix entered by user.
   for(i = 0; i < r; ++i)
     for(j = 0; j < c; ++j)
     {
        cout << "Enter element a" << i + 1 << j + 1 << " : ";
        cin >> a[i][j];
```

```
   }
// Storing elements of second matrix entered by user.
cout << endl << "Enter elements of 2nd matrix: " << endl;
for(i = 0; i < r; ++i)
  for(j = 0; j < c; ++j)
  {
     cout << "Enter element b" << i + 1 << j + 1 << " : ";
     cin >> b[i][j];
  }
// Adding Two matrices
for(i = 0; i < r; ++i)
   for(j = 0; j < c; ++j)
      sum[i][j] = a[i][j] + b[i][j];
// Displaying the resultant sum matrix.
cout << endl << "Sum of two matrix is: " << endl;
for(i = 0; i < r; ++i)
   for(j = 0; j < c; ++j)
   {
      cout << sum[i][j] << "  ";
      if(j == c - 1)
         cout << endl;
   }
   // Subtracting Two matrices
for(i = 0; i < r; i++)
   for(j = 0; j < c; j++)
     diff[i][j] = a[i][j] - b[i][j];

// Displaying the resultant sum matrix.
cout << endl << "Difference of two matrix is: " << endl;
for(i = 0; i < r;i++)
   for(j = 0; j < c; j++)
```

```
        {
            cout << diff[i][j] << "  ";
            if(j == c + 1)
                cout << endl;
        }

    return 0;
}
```

5. **Demonstrate simple inheritance concept by creating a base class FATHER with data members: *First Name, Surname, DOB & bank Balance* and creating a derived class SON, which inherits: Surname & Bank Balance feature from base class but provides its own feature: First Name & DOB. Create & initialize F1 & S1 objects with appropriate constructors & display the FATHER & SON details.**

```
#include <iostram>
#include<string>
using namespace std;
class FATHER {
protected:
 string Surname;
double BankBalance; p
ublic: FATHER(const string& surname, double balance) : Surname(surname), BankBalance(balance) {}
void display() {
cout << "Surname: " << Surname << endl;
 cout << "Bank Balance: " << BankBalance << endl; }
};
class SON : public FATHER {
 private:
string FirstName;
string DOB;
```

```
 public:
SON(const string& firstname, const string& surname, const string& dob, double balance) : FATHER(surname, balance), FirstName(firstname), DOB(dob) {}
void display() {
cout << "First Name: " << FirstName << endl;
FATHER::display();
cout << "DOB: " << DOB << endl;
 }
 };

int main() {
 FATHER F1("Smith", 10000.0);
 SON S1("John", "Smith", "2000-01-01", 5000.0);
cout << "Father's Details: " << endl;
 F1.display(); cout << endl; cout << "Son's Details: " << endl;
S1.display();
 return 0;
}
```

Output:

```
Father's Details:
Surname: Smith
Bank Balance: 10000

Son's Details:
First Name: John
Surname: Smith
Bank Balance: 5000
DOB: 2000-01-01
```

6. Write a C++ program to define class name FATHER & SON that holds the income respectively. Calculate & display total income of a family using Friendfunction.

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

class SON;
class FATHER {
private:
double income;
public:
   FATHER(double income):income(income)
{}
friend double getTotalIncome(const FATHER&father,const SON&son);
};

class SON{
private:
double income;
public:
SON(double income):income(income){}
//friend function declaration to access FATHER's income
friend double getTotalIncome(const FATHER&father,const SON&son);
};
//friend function definition to calculate total income
double getTotalIncome(const FATHER&father,const SON&son){
return father.income+son.income;}

int main()
{
```

```
double fatherIncome,sonIncome;
cout<<"Enter Father's Income:";
cin>>fatherIncome;
cout<<"Enter son's income:";
cin>>sonIncome;
FATHER father(fatherIncome);
SON son(sonIncome);
double totalIncome=getTotalIncome(father,son);
cout<<"Total income of the family:"<<totalIncome<<endl;
return 0;
}
```

Output:

```
Enter Father's Income:150000
Enter son's income:200000
Total income of the family:350000
```

7. Write a C++ program to accept the student detail such as name & 3 different marks by get_data()method & display the name & average of marks using display() method. Define a friend function for calculating the average marks using the method mark_avg().

```
#include <iostream>
#include <string>
Using namespace std;
class Student {
private:
   string name;
   double marks[3];

public:
```

```
    void get_data() {
        cout << "Enter student's name: ";
         cin >> name;
      for (int i = 0; i < 3; i++) {
        cout << "Enter mark " << i + 1 << ": ";
        cin >> marks[i];
      }
    }

    friend double mark_avg(const Student& student); // Friend function for average
calculation

    void display() {
        cout << "Student's name: " << name << std::endl;
      for (int i = 0; i < 3; i++) {
        cout << "Mark " << i + 1 << ": " << marks[i] << std::endl;
      }
      cout << "Average marks: " << mark_avg(*this) << std::endl;
    }
};

// Friend function to calculate average marks
double mark_avg(const Student& student) {
    double total = 0.0;
    for (int i = 0; i < 3; i++) {
      total += student.marks[i];
    }
    return total / 3.0;
}
int main() {
    Student student;
```

```
    student.get_data();
    student.display();
    return 0;
}
```

Output:

Enter student's name: Nikhitha
Enter mark 1: 80
Enter mark 2: 75
Enter mark 3: 90
Student's name: Nikhitha
Mark 1: 80
Mark 2: 75
Mark 3: 90
Average marks: 81.6667

8. Write a C++ program to explain virtual function (Polymorphism) by creating a base class polygon which has virtual function areas two classes rectangle & triangle derived from polygon & they have area to calculate & return the area of rectangle & triangle respectively.

```
#include <iostream>
using namespace std;
class Polygon {
public:
virtual double area() = 0; // Pure virtual function
virtual ~Polygon() {} // Virtual destructor
};
class Rectangle : public Polygon {
private:
```

```
double  width;
double height;
public:
Rectangle(double w, double h) : width(w), height(h) {}
double area() override {
return width * height;
}
};
class Triangle : public Polygon {
private:
double base;
double height;
public:
Triangle(double b, double h) : base(b), height(h) {}
double area() override {
return 0.5 * base * height;
}
};
int main() {
Rectangle rectangle(5, 3);
Triangle triangle(4, 6);
Polygon* poly1 = &rectangle;
Polygon* poly2 = &triangle;
cout << "Area of rectangle: " << poly1->area() << endl;
cout << "Area of triangle: " << poly2->area() << endl;
return 0;
}
```

Output:

Area of rectangle: 15

Area of triangle: 12

9 Design, develop and execute a program in C++ based on the following requirements: An EMPLOYEE class containing data members & members functions: i) Data members: employee number (an integer), Employee_ Name (a string of characters), Basic_ Salary (in integer), All_ Allowances (an integer), Net_Salary (an integer). (ii) Member functions: To read the data of an employee, to calculate Net_Salary & to print the values of all the data members. (All_Allowances = 123% of Basic, Income Tax (IT) =30% of gross salary (=basic_ Salary_All_Allowances_IT).

```
#include <iostream>
#include <string>
using namespace std;
class EMPLOYEE {
private:
   int employeeNumber;
   string employeeName;
   int basicSalary;
   int allAllowances;
   int netSalary;

public:
   // Member function to read employee data
   void readData() {
      cout << "Enter Employee Number: ";
      cin >> employeeNumber;
      cin.ignore();  // Clear the newline character from the buffer
      cout << "Enter Employee Name: ";
      getline(std::cin, employeeName);
      cout << "Enter Basic Salary: ";
         cin >> basicSalary;
   }
```

```
    // Member function to calculate Net Salary
    void calculateNetSalary() {
        allAllowances = (int)(1.23 * basicSalary);
        int grossSalary = basicSalary + allAllowances;
        int incomeTax = (int)(0.3 * grossSalary);
        netSalary = grossSalary - incomeTax;
    }

    // Member function to print employee details
    void printData() {
            cout << "Employee Number: " << employeeNumber << std::endl;
              cout << "Employee Name: " << employeeName << std::endl;
            cout << "Basic Salary: " << basicSalary << std::endl;
             cout << "All Allowances: " << allAllowances << std::endl;
            cout << "Net Salary: " << netSalary << std::endl;
    }
};

int main() {
    EMPLOYEE employee;

    // Read employee data
    employee.readData();

    // Calculate net salary
    employee.calculateNetSalary();

    // Print employee details
    std::cout << "\nEmployee Details:" << std::endl;
    employee.printData();
```

```
    return 0;
}
```

Output:

```
Enter Employee Number: 133
Enter Employee Name: John
Enter Basic Salary: 5000
Employee Details:
Employee Number: 133
Employee Name: John
Basic Salary: 5000
All Allowances: 6150
Net Salary: 7805
```

10. Write a C++ program with different class related through multiple inheritance & demonstrate the use of different access specified by means of members variables & members functions.

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

class Shape {
public:
double area;
Shape():area(0.4){}
void calculateArea(){
cout<<"calculating area in shape class"<<endl;
}
};
```

```
class Color{
protected:
string color;
public:
Color():color("No color"){}
void setColor (string newcolor){
color=newcolor;
}
};
class ColoredShape:public Shape,public Color{
public:
ColoredShape(){}
void displayColor(){
cout<<"color:"<<color<<endl;
}
};
int main() {
ColoredShape cs;
// Accessing members of Shape class
//cs.calculateArea();
cout << "Area: " << cs.area << endl;
// Accessing members of Color class (through ColoredShape)
cs.setColor("Red");
cs.displayColor();
return 0;
}
```

Output:

Calculating area in Shape class

Area: 0

Color: Red

11. Write a C++ program to create three objects for a class named count object with data members such as roll_no & Name. Create a members function set_data ( ) for setting the data values & display ( ) member function to display which object has invoked it using "this" pointer.

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

class CountObject {
private:
int roll_no;
string Name;
public:
void set_data(int roll, const string& name) {
roll_no = roll;
Name = name;
}
void display() {
cout << "Roll Number: " << roll_no << endl;
cout << "Name: " << Name << endl;
cout << "This object is invoked!" << endl;
cout << "Object Address: " << this << endl;
}
};

int main() {
CountObject obj1, obj2, obj3;
obj1.set_data(1, "John");
obj2.set_data(2, "Jane");
obj3.set_data(3, "Alice");
cout << "Object 1 details:" << endl;
```

```
obj1.display();
cout << endl;
cout << "Object 2 details:" << endl;
obj2.display();
cout << endl;
cout << "Object 3 details:" << endl;
obj3.display();
cout << endl;
return 0;
}
```

Output:

```
Object 1 detail:
Roll Number: 1
Name: John
This object is invoked!
Object Address: 0x7ffd07e28ae0

Object 2 details:
Roll Number: 2
Name: Jane
This object is invoked!
Object Address: 0x7ffd07e28ab0

Object 3 details:
Roll Number: 3
Name: Alice
This object is invoked!
Object Address: 0x7ffd07e28a80
```

12. Write a C++ program to implement exception handling with minimum 5 exceptions classes including two built in exceptions.

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

class Invalid_argument : public exception {
public:
   const char* what() const throw(){
      return " Invalid_argument: Division by zero";
   }
};

class Out_of_range : public exception {
public:
   const char* what() const throw(){
      return " Out_of_range: Negative value";
   }
};

class newException1 : public exception {
public:
   const char* what() const throw() {
      return " newException1: Something went wrong”;
   }
};

class newException2 : public exception {
public:
   const char* what() const throw() {
      return " newException2: Another issue occurred!";
   }
};

class newException3 : public exception {public:
   const char* what() const throw() {
```

```
        return " newException3: overflow”;
    }
};

class newException4 : public exception {public:
    const char* what() const throw() {
        return " newException4: Yet another problem!";
    }
};
class newException5 : public exception {public:
    const char* what() const throw() {
        return " newException5: special character are entered;
    }
};

// Function that throws exceptions
void performOperation(int value) {
    if (value == 0) {
        throw Invalid_argument();
```

```
}else if (value < 0) {
        throw Out_of_range();
    } else if (value == 1) {
        throw newException1();
    } else if (value == 2) {
        throw newException2();
    } else if (value == 3) {
        throw newException3();
    }
    else if(value==4){
         throw newException4();}
    else if(value>4){
         throw newException5();
  }
}

 int main() {
    try {
        int userInput;
        cout << "Enter a value (0 to 4): ";
       cin >> userInput;

        performOperation(userInput);

    } catch (const Invalid_argument& e) {
        cerr << "Exception caught: " << e.what() << endl;

    } catch (const Out_of_range& e) {
        cerr << "Exception caught: " << e.what() << endl;

    } catch (const newException1& e) {
       cerr << "Exception caught: " << e.what() << endl;
```

```
} catch (const newException2 & e) {
    cerr << "Exception caught: " << e.what() << endl;
```

```
    } catch (const newException3& e) {
        cerr << "Exception caught: " << e.what() << endl;

    } catch (const newException4& e) {
        cerr << "Unknown exception caught: " << e.what() <<endl;
    }
    catch (const newException5& e) {
        cerr << " exception caught but a special character " << e.what() <<endl;
    }

    return 0;
}
```

Output 1:

Enter a value (0 to 4): 0

Exception caught: Invalid argument: Division by zero

Output 2: Enter a value (0 to 4): 1

Exception caught: newException1: Something went wrong

Output 3: Enter a value (0 to 4): 2

Exception caught: newException2: Another issue occurred!

Output 4: Enter a value (0 to 4): 3

Exception caught: newException3: overflow

Output 5: Enter a value (0 to 4): 4

Exception caught: newException4: Yet another problem

Output 6: Enter a value (0 to 4): 8

Exception caught: newException5:: special character to be entered

Output 7: Enter a value (0 to 4): -8

Exception caught:Out_of_range:Negative value