|  |  |
| --- | --- |
| **Name** | ADITI RAO |
| **UID no.** | 202220003 |
| **Experiment No.** | 4 |

|  |  |
| --- | --- |
| **AIM:** | **Program on Polymorphism: Implement a Program to demonstrate method overloading** |
| **Program 1** | |
| **PROBLEM STATEMENT:** | Create a Date class with data int year, int month, int date, int hrs, int min, int sec. Create a default, no-argument constructor which sets the default date to January 1, 2000, 00:00:00 Create 3 overloaded setter methods  void setDate(int year, int month, int date)  void setDate(int year, int month, int date, int hrs, int min)  void setDate(int year, int month, int date, int hrs, int min, int sec)  Also write a displayDate() method which will display the date depending on the type of date object created. |
| **ALGORITHM:** | 1. CLASS Date: 2. PRIVATE:   year, month, day, hrs, mins, sec: integers  PUBLIC:   1. Date(): constructor 2. year = 2000 3. month = 1 4. day = 1 5. hrs = 0 6. mins = 0 7. sec = 0 8. VOID setDate(y: int, m: int, d: int): 9. year = y 10. month = m 11. day = d 12. hrs = 0 13. mins = 0 14. sec = 0 15. VOID setDate(y: int, m: int, d: int, h: int, mi: int): void 16. year = y 17. month = m 18. day = d 19. hrs = h 20. mins = mi 21. sec = 0 22. VOID setDate(y: int, m: int, d: int, h: int, mi: int, s: int): 23. year = y 24. month = m 25. day = d 26. hrs = h 27. mins = mi 28. sec = s 29. displayDate(): void 30. if hrs == 0 and mins == 0 and sec == 0:   print day, "-", month, "-", year, "\n"   1. else if sec == 0:   print day, "-", month, "-", year, " ", hrs, ":", mins, "\n"   1. else:   print day, "-", month, "-", year, " ", hrs, ":", mins, ":", sec, "\n"   1. main(): 2. print "This program was brought to you by Aditi Rao, 2022200003. \n" 3. newDate = Date() 4. print "You are to choose the method of date presentation. \n1. To show year, month and day \n2. To show hours and minutes as well \n3. To show seconds as well.\n" 5. read sw\_var 6. switch sw\_var: 7. case 1: 8. read year, month, day 9. newDate.setDate(year, month, day) 10. newDate.displayDate() 11. break 12. case 2: 13. read year, month, day, hrs, mins 14. newDate.setDate(year, month, day, hrs, mins) 15. newDate.displayDate() 16. break 17. case 3: 18. read year, month, day, hrs, mins, sec 19. newDate.setDate(year, month, day, hrs, mins, sec) 20. newDate.displayDate() 21. break 22. default:   print "Invalid choice"   1. END |
| **PROGRAM:** | #include <iostream>  using namespace std;  class Date  {  private:  int year, month, day, hrs, mins, sec;  public:  // Default constructor  Date()  {  year = 2000;  month = 1;  day = 1;  hrs = 0;  mins = 0;  sec = 0;  }  // Setter method for date with only year, month and date  void setDate(int y, int m, int d)  {  year = y;  month = m;  day = d;  hrs = 0;  mins = 0;  sec = 0;  }  // Setter method for date and time without seconds  void setDate(int y, int m, int d, int h, int mi)  {  year = y;  month = m;  day = d;  hrs = h;  mins = mi;  sec = 0;  }  // Setter method for date and time with seconds  void setDate(int y, int m, int d, int h, int mi, int s)  {  year = y;  month = m;  day = d;  hrs = h;  mins = mi;  sec = s;  }  // Display method to display date and time  void displayDate()  {  if (hrs == 0 && mins == 0 && sec == 0)  cout << "\n" << day << "-" << month << "-" << year << " " << endl;  else if (sec == 0)  cout << "\n" << day << "-" << month << "-" << year << " " << hrs << ":" << mins << endl;  else  cout << "\n" << day << "-" << month << "-" << year << " " << hrs << ":" << mins << ":" << sec << endl;  }  };  int main()  {  cout << "This program was brought to you by Aditi Rao, 2022200003. \n" << endl;  Date newDate;  cout << "You are to choose the method of date presentation. \n1. To show year, month and day \n2. To show hours and minutes as well \n3. To show seconds as well.\n" << endl;  int sw\_var;  cin >> sw\_var;  switch (sw\_var)  {  case 1:  {  int year, month, day;  cout << "\nEnter year: ";  cin >> year;  cout << "Enter month: ";  cin >> month;  cout << "Enter day: ";  cin >> day;  newDate.setDate(year, month, day);  newDate.displayDate();  break;  }  case 2:  {  int year, month, day, hrs, mins;  cout << "\nEnter year: ";  cin >> year;  cout << "Enter month: ";  cin >> month;  cout << "Enter day: ";  cin >> day;  cout << "Enter hours: ";  cin >> hrs;  cout << "Enter minutes: ";  cin >> mins;  newDate.setDate(year, month, day, hrs, mins);  newDate.displayDate();  break;  }  case 3:  {  int year, month, day, hrs, mins, sec;  cout << "\nEnter year: ";  cin >> year;  cout << "Enter month: ";  cin >> month;  cout << "Enter day: ";  cin >> day;  cout << "Enter hours: ";  cin >> hrs;  cout << "Enter minutes: ";  cin >> mins;  cout << "Enter seconds: ";  cin >> sec;  newDate.setDate(year, month, day, hrs, mins, sec);  newDate.displayDate();  break;  }  default:  cout << "\nInvalid choice" << endl;  }  return 0;  } |
| **RESULT:** |  |
| **Program 2** | |
| **PROBLEM STATEMENT:** | Write a C++ program to implement method overloading by implementing the Power class which contains a function raisePower() to raise a number a b in the following different forms .   1. The function takes a DOUBLE value for a and INT value for b.   Use a default value of 2 for b to make the function to calculate squares when this argument is omitted.   1. Implement another function that performs the same operations but takes INT for both a and b. |
| **ALGORITHM:** | START  private:  double num1;  int num2;  public:  Power()  num1 = 0.0;  num2 = 0;  ENDCONSTRUCTOR    void raisePower(double a, int b = 2)  double result = 1;  for(int i = 0; i < b; i++)  result \*= a;  num1 = result;  num2 = b;  cout << "The result is " << result << endl;  ENDFUNCTION    void raisePower(int a, int b)  double result = 1;  for(int i = 0; i < b; i++)  result \*= a;  num1 = result;  num2 = b;  cout << "The result is " << result << endl;  ENDFUNCTION    void display()  cout << "num1 = " << num1 << endl;  cout << "num2 = " << num2 << endl;  ENDFUNCTION  ENDCLASS  Main()  MAKE obj\_pow;  // Testing the function with integer arguments  PRINT obj\_pow.raisePower(2, 3)  // Testing the function with double and integer arguments  PRINT\_pow.raisePower(2.5)  PRINT << obj\_pow.raisePower(2.5, 3)  ENDMAIN  END |
| **PROGRAM:** | #include <iostream>  #include <cmath>  using namespace std;  class Power  {  public:  int raisePower(int a, int b)  {  return pow(a, b);  }  double raisePower(double a, int b = 2)  {  return pow(a, b);  }  };  int main()  {  Power obj\_pow;  // Testing the function with integer arguments  cout << "2 raised to the power of 3: " << obj\_pow.raisePower(2, 3) << endl;  // Testing the function with double and integer arguments  cout << "2.5 raised to the power of 2: " << obj\_pow.raisePower(2.5) << endl;  cout << "2.5 raised to the power of 3: " << obj\_pow.raisePower(2.5, 3) << endl;  return 0;  } |
| **RESULT:** |  |
| **Program 3** | |
| **PROBLEM STATEMENT:** | Write a program that overloads absolute 3 functions namely absolute that return absolute values of int, floating point number and double. |
| **ALGORITHM:** | CLASS Power:  FUNCTION raisePower(a, b = 2):  RETURN a^b  ENDFUNCTION  ENDCLASS  obj\_pow = Power()  PRINT "2 raised to the power of 3: " + obj\_pow.raisePower(2, 3)  PRINT "2.5 raised to the power of 2: " + obj\_pow.raisePower(2.5)  PRINT "2.5 raised to the power of 3: " + obj\_pow.raisePower(2.5, 3)  ENDMAIN |
| **PROGRAM:** | #include <iostream>  #include <cmath>  using namespace std;  class Power  {  public:  int raisePower(int a, int b)  {  return pow(a, b);  }  double raisePower(double a, int b = 2)  {  return pow(a, b);  }  };  int main()  {  Power obj\_pow;  // Testing the function with integer arguments  cout << "2 raised to the power of 3: " << obj\_pow.raisePower(2, 3) << endl;  // Testing the function with double and integer arguments  cout << "2.5 raised to the power of 2: " << obj\_pow.raisePower(2.5) << endl;  cout << "2.5 raised to the power of 3: " << obj\_pow.raisePower(2.5, 3) << endl;  return 0;  } |
| **RESULT:** |  |
| **CONCLUSION** | Polymorphism is a key concept in object-oriented programming that allows different objects to be treated as if they are of the same type. Method overloading is one form of polymorphism in which multiple methods can have the same name but different parameters. In this program, we have demonstrated method overloading in C++ by defining multiple functions with the same name but different parameter lists. By doing so, we can reuse the function name to perform similar operations on different data types or with different argument lists. Method overloading is a powerful feature of C++ that can make code more concise and easier to read, while also providing flexibility and extensibility. |