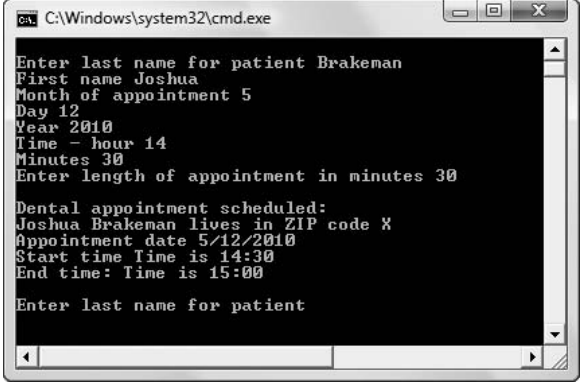
|  |  |  |
| --- | --- | --- |
| |  | | --- | | Lab02 Class and Template | |  | |
| Exercise 1: Constructors (20 min) |

|  |
| --- |
| Reuse the Point class and Triangle class in Lab 01, and use the new main function provided below. The new main function does not work.   1. Why the new main function does not work? 2. If you write an overloaded constructor only, the program does not work because C++ compiler won’t provide a default constructor once we write an overloaded constructor. So the best solution is to write a single constructor that acts as both default constructor and overloaded constructor. Write code to achieve it. Initialize all numbers 0 in default constructor.   int main() {  Point p[] = { Point(), Point(1), Point(2,3) };  for (int i = 0; i < 3; i++)  cout << "(" << p[i].getX() << "," << p[i].getY() << ")\n";  cout << "\nTriangle ";  Triangle t;  t.display();  }  Expected Output:  (0,0)  (1,0)  (2,3)  Triangle Points:  (0,0)  (0,0)  (0,0)  Perimeter = 0 |
| Exercise 2: Vector (50 min) |
| Use the Point class and Triangle class in Exercise 1, write a program that asks the user to enter many points, then ask the user to select any 3 points to form triangles. The program shall record down all the points and triangles created by the user. Add the following member and use it in your main function.  Triangle:: Triangle (Point&, Point&, Point&); |

|  |
| --- |
| Exercise 3: Function Template (20 min) |
| The following program is given. The program fails to compile because only the int version of function find and function print are provided. char version is not provided. Instead of writing a char version of function find and function print, a better is to write a template version of function find and function print since the only difference between the int version and char version is data type. Write code to accomplish that.  #include <iostream>  #include <string>  #include <vector>  using namespace std;  int find (vector<int>& v, int target) {  for (int i = 0; i < v.size(); i++)  if (v[i] == target)  return i; // Target found at index i.  return -1; // Target not found.  }  void print (vector<int>& v) {  for (int i = 0; i < v.size(); i++)  cout << v[i] << " ";  cout << endl;  }  int main() {  int intTarget;  vector<int> intv;  intv.push\_back (33);  intv.push\_back (55);  intv.push\_back (11);  print (intv);  cout << "Enter target: ";  cin >> intTarget;  int i = find (intv, intTarget);  if (i >= 0)  cout << "Target found at index " << i << ".\n";  else  cout << "Target not found.";  char charTarget;  vector<char> charv;  charv.push\_back ('C');  charv.push\_back ('A');  charv.push\_back ('B');  print (charv);  cout << "Enter target: ";  cin >> charTarget;  i = find (charv, charTarget);  if (i >= 0)  cout << "Target found at index " << i << ".\n";  else  cout << "Target not found.";  } |
| Exercise 4: Class Template (20 min) |
| The following program is given. The class is very rigid in terms of the data type of the attributes. Covert the class to make the data type of the attributes flexible. Test your new class.  //<Use Function Template>  class Box {  int a;  char b;  string c;  public:  Box (int a, char b, string c) : a(a), b(b), c(c) {}  int getA() { return a; }  char getB() { return b; }  string getC() { return c; }  };  int main() {  Box p1 (1, 'A', "Apple");  cout << p1.getA() << " " << p1.getB() << " "  << p1.getC() << endl;  Box\* p2 = new Box(2, 'B', "Boy");  cout << p2->getA() << " " << p2->getB() << " "  << p2->getC() << endl;  delete p2;  } |

|  |
| --- |
| Take Home Exercises |

* 1. Write the class definition for a Date class that contains three integer data members: month, day, and year. Include a default constructor that assigns the date 1/1/2000 to any new object that does not receive arguments. Also include a function that displays the Date object. Write a driver program (that contains the main() function) in which you instantiate two Date objects—one that you create using the default constructor values, and one that you create using three arguments—and display its values.
  2. Create a class named Time that contains integer fields for hours and minutes. Store the hours in military time, that is, 0 through 23. Add a function that displays the fields, using a colon to separate hours and minutes. (Make sure the minutes display as two digits. For example, 3 o’clock should display as 3:00, not 3:0.) Add another function that takes an argument that represents minutes to add to the time. The function updates the time based on the number of minutes added. For example, 12:30 plus 15 is 12:45, 14:50 plus 20 is 15:10, and 23:59 plus 2 is 0:01. The Time constructor requires an argument for hours. The argument for minutes is optional; the value defaults to 0 if no argument is supplied. The constructor ensures that the hours field is not greater than 23 and that the minutes field is not greater than 59; default to these maximum values if the arguments to the constructor are out of range. Write a driver program that instantiates an array of at least four Time objects and demonstrates that they display correctly both before and after varying amounts of time have been added to them.
  3. Create a Person class that includes fields for last name, first name, and zip code. Include a default constructor that initializes last name, first name, and zip code to “X” if no arguments are supplied. Also include a display function. Write a driver program that instantiates and displays two Person objects: one that uses the default values, and one for which you supply your own values.
  4. Create a class named DentalAppointment. Include fields for a patient’s data (use the Person class from Exercise 3), the date (using the Date class from Exercise 1), the time (use the Time class from Exercise 2), and the duration of the appointment in minutes. Also include a field that contains the ending time of the appointment; this field will be calculated based on the start time and the duration using the Time class function that adds minutes to a Time object. The DentalAppointment constructor requires a first and last name, and a month, day, year, hour, and minute for the appointment. Allow a DentalAppointment to be constructed with or without an additional argument for appointment duration, and force the duration to 30 minutes when no argument is supplied. The constructor does not allow any appointment over 240 minutes. The constructor calculates the appointment ending time based on the start time and the duration. Also include a display function for the DentalAppointment class. Write a driver program that loops at least three times, prompting the user for DentalAppointment data and displaying all the information. (Note, if you use the Person class display function, the zip code will be “X”; this is acceptable.). The following figure shows an example output of the program:



* 1. Create a class named MagazineSubscription. Include fields for the subscriber (use the Person class you created in Exercise 3) and the subscription’s start and expiration dates (use the Date class you created in Exercise 1). Include a constructor that takes three arguments—a Person and two Dates. Also include a display function that displays MagazineSubscription fields by calling the Person and Date display functions. Write a driver program in which you instantiate a Person object and two Date objects. Use these as arguments to the constructor that instantiates a MagazineSubscription object. Display the MagazineSubscription object.
  2. Define a class named CoffeeOrder. Declare a private static field that holds the price of a cup of coffee as $1.25. Include private integer fields that you set to a flag value of 1 or 0 to indicate whether the order should have any of the following: cream, milk, sugar, or artificial sweetener. Include a public function that takes a user’s order from the key board and sets the values of the four fields in response to four prompts. If the user indicates both milk and cream, turn off the milk flag to allow only cream. If the user indicates both sugar and artificial sweetener, turn off the artificial sweetener flag, allowing only sugar. Include another function that displays the user’s completed order. Write a driver program to test the functions of the class

Using the CoffeeOrder class, write a driver program that continues to ask a user for an order in a loop until the user indicates the order is complete or 10 orders have been placed, whichever comes first. After the user indicates that ordering is complete, display a recap of all the coffee orders, including the cream, milk, sugar, and sweetener status of each, as well as a count of the number of coffees ordered and the total price.