|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | Lab03 Inheritance | |  | | |
| Exercise 1: Basic Inheritance (40 min) | |
| The following UML Class Diagram is given.   |  |  |  | | --- | --- | --- | |  | Human |  | |  | - name: string |  | |  | + Human (name: string)  + getName(): string  + setName(name: string):  void |  | |  |  |  | | Student |  | Employee | | - id: int |  | - salary: double | | + Student (name: string,  id: int)  + getId(): int  + setId(id: int): void |  | + Employee (name: string,  salary: double)  + getSalary(): double  + setSalary(salary: double):  void |  1. Write a complete definition for Human class. Add a message "A human is created: Name = xxx" in Human constructor. Write a main function to test the Human class. 2. Repeat Step 1 for Student class. Add a message "A student is created: Name = xxx. Id = yyy" in Student constructor. Add a Student object in your main function to test the Student class. Note how the Student constructor invokes the Human constructor. 3. Repeat Step 2 for Employee class. Add a message "An employee is created: Name = xxx. Salary = yyy" in Employee constructor. Add a Employee object in your main function to test the Employee class. | |

|  |
| --- |
| Exercise 2: Multiple Inheritance, Diamond Inheritance (30 min) |
| A new class WorkingStudent has been added to our UML Class Diagram. A working student is a student and is also an employee at the same time. Multiple inheritances is used to model the working student.   |  |  |  | | --- | --- | --- | |  | Human |  | |  | - name: string |  | |  | + Human (name: string)  + getName(): string  + setName(name: string):  void |  | |  |  |  | | Student |  | Employee | | - id: int |  | - salary: double | | + Student (name: string,  id: int)  + getId(): int  + setId(id: int): void |  | + Employee (name: string,  salary: double)  + getSalary(): double  + setSalary(salary: double):  void | |  |  |  | |  |  |  | |  | WorkingStudent |  | |  |  |  | |  | + WorkingStudent (name: string,  id: int, salary, double)  + print(): void |  |  * 1. Diamond inheritance occurs in this case so a WorkingStudent object will inherit 2 sets of members from Human object. Hence, to use a Human member from WorkingStudent, we must specify exactly whether we want to use it from Student or Employee.   2. Write a complete definition for WorkingStudent class. Add a WorkingStudent object in your main function to test the WorkingStudent class. The constructor for WorkingStudent is given below:   WorkingStudent (string name, int id, double salary)  : Student(name, id), Employee (name, salary) {  cout << "A working student is created: Name = " << Student::getName()  << ". Id = " << getId() << ". Salary" << getSalary() << endl;  }   1. Compile and run your program. Notice that a Human constructor is called 2 times when creating a WorkingStudent object. 2. Is it desirable to inherit 2 sets of Human members in a WorkingStudentobject? Why? |
| Exercise 3: Virtual Inheritance (20 min) |
| Update your classes in Exercise 3 to use virtual inheritance so that there is only one set of Human members in a WorkingStudent object. The constructor for WorkingStudent has to be updated and is given below:  WorkingStudent (string name, int id, double salary)  : Human (name), Student(name, id), Employee (name, salary) {  cout << "A working student is created: Name = " << getName()  << ". Id = " << getId() << ". Salary" << getSalary() << endl;  } |

|  |
| --- |
| Exercise 4: protected Access Privilege (20 min) |
| The attributes of all classes have been updated to use protected access privilege as shown in the UML Class Diagram below.   |  |  |  | | --- | --- | --- | |  | Human |  | |  | # name: string |  | |  | + Human (name: string)  + getName(): string  + setName(name: string):  void |  | |  |  |  | | Student |  | Employee | | # id: int |  | # salary: double | | + Student (name: string,  id: int)  + getId(): int  + setId(id: int): void |  | + Employee (name: string,  salary: double)  + getSalary(): double  + setSalary(salary: double):  void | |  |  |  | |  |  |  | |  | WorkingStudent |  | |  |  |  | |  | + WorkingStudent (name: string,  id: int, salary, double)  + print(): void |  |  1. If you just update the attributes' access privilege, your program in Exercise 3 and 4 should still work fine. Try it. 2. We were previously prevented from directly access the attributes in the WorkingStudent's print method because the attributes were private. Now with protected privilege we can directly access the attributes in the print method. Try it. |

|  |
| --- |
| Take Home Exercises |
| * 1. Create a base class named Book. Data fields include title and author; functions include those that can set and display the fields. Derive two classes from the Book class: Fiction, which also contains a numeric grade reading level, and NonFiction, which contains a variable to hold the number of pages. The functions that set and display data field values for the subclasses should call the appropriate parent class functions to set and display the common fields, and include specific code pertaining to the new subclass fields. Write a driver program that demonstrates the use of the classes and their functions.   2. Below is the class declaration for the Person class.   class Person  {  public:  Person();  Person(string name);  Person(Person& the\_object);  void set\_name(string name);  string get\_name();  private:  string name;  };  Provide the implementation for the Person class above. Create a base class called Vehicle that has the manufacturer’s name (type string), number of cylinders in the engine (type *int*), and owner (type Person). Then create a class called Truck that is derived from Vehicle and has additional properties: load capacity in tons (type *double* since it may contain a fractional part) and towing capacity in pounds (type *int*). Be sure your classes have a reasonable complement of constructors, accessor, and mutator member functions, an overloaded assignment operator and a copy constructor, Write a driver program that tests all your member functions.   * 1. Implement a base class Account and derived classes Savings and Checking. In the base class, supply member functions deposit and withdraw. Provide a function daily\_interest that computes and adds the daily interest. For calculations, assume that every month has 30 days. Checking accounts yield interest of 3 percent monthly on balances over $1000. Savings accounts yield interest of 6 percent on the entire balance. Write a driver program that makes a month’s worth of deposits and withdrawals and calculates the interest every day.   2. Create a RestaurantMeal class that holds the name and price of a food item served by a restaurant. Its constructor requires arguments for each field. Create a HotelService class that holds the name of the service, the service fee, and the room number to which the service was supplied. Its constructor also requires arguments for each field. Create a RoomServiceMeal class that inherits from both RestaurantMeal and HotelService. Whenever you create a RoomServiceMeal object, the constructor assigns the string “room service” to the name of the service field, and $4.00 is assigned to the service fee inherited from HotelService. Include a RoomServiceMeal function that displays all of the fields in a RoomServiceMeal by calling display functions from the two parent classes. Additionally, the display function should display the total of the meals plus the room service fee. Write a driver program that instantiate a RoomServiceMeal object that inherits from both classes. For example, a “steak dinner” costing $19.99 is a “room service” provided to room 1202 for a $4.00 fee. |