# 

# SAVITCH ABSOLUTE C++

# Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Term # \_\_\_\_

**Homework 9 – On Lecture 11 - UML to C++**

**(100 points) Hours:**

**HARCOPIES: The homework is to be turned in as a *PAPER AND PENCIL i.e., HANDWRITTEN ANSWER ONLY!(with your terminal #!)* in the first ten minutes of the due date class.**

**TURNING IN THE HOMEWORK INSTRUCTIONS will be PENALTY OF -10 points.**

**I UNDERSTAND THAT TURNING ANOTHER’s WORK IN is CHEATING.**

**I UNDERSTAND THAT ANY KIND OF DISSEMINATION of this WORK is CHEATING.**

**I CERTIFY THAT THE HOMEWORKs SOLUTIONs ARE MY OWN WORK!**

**?**

**X**

**V**

**SIGNATURE:**

**HOMEWORK CHECKLIST (YOU MUST GRADE YOURSELF!):**

1. **DID TURN IN HOMEWORK INSTRUCTIONS? \* -10 points**

**1.? 10 points**

**2.? 10 points**

**3.? 10 points**

**TA check, is Homework9.doc**

**in BB?**

**TA check, is Homework1.doc**

**in BB?**

**TA check, is Homework1.doc**

**in BB?**

**4.? 10 points**

**5.? 30 points**

**6.? 10 points**

**7.? 10 points**

**8.? 10 points**

**\* If NOT, do not enter anything in the box!**

TA **grade or check**

**PLEASE ENTER YOUR GRADE IN THIS BOX:**

**By Hand?**

**1.** (10 pts) **UML Class Diagram** (**MICROSOFT WORD; Textual Analysis – TA Cut&Paste&Rearrange**). **ClockType**

Define a Class to implement the time of day. Because a clock gives the time of day, let us call this Class ClockType. Furthermore, to represent time in computer memory, we use three int variables: one to represent the hours **hr**, one to represent the minutes **min**, and one to represent the seconds **sec**.

ClockType: sec:int //3

ClockType: min:int //2

ClockType: hr:int //1

ClockType

We also want to perform the following operations on the time:

ClockType: setTime(int): void //1

1. Set the time **setTime**.

ClockType: getTime():int //2

2. Retrieve the time **getTime**.

ClockType: printTime(): void //3

3. Print the time **printTime**.

ClockType: incrementSeconds(): void //4

4. Increment the time by one second **incrementSeconds**.

5. Increment the time by one minute **incrementMinutes**.

ClockType: incrementMinutes(): void //5

6. Increment the time by one hour **incrementHours**.

ClockType: incrementHours(): void //6

7. Compare the two times for equality **equalTime**.

ClockType: equalTime(ClockType): void //7

**Draw the UML “compilable” Class Diagram.**

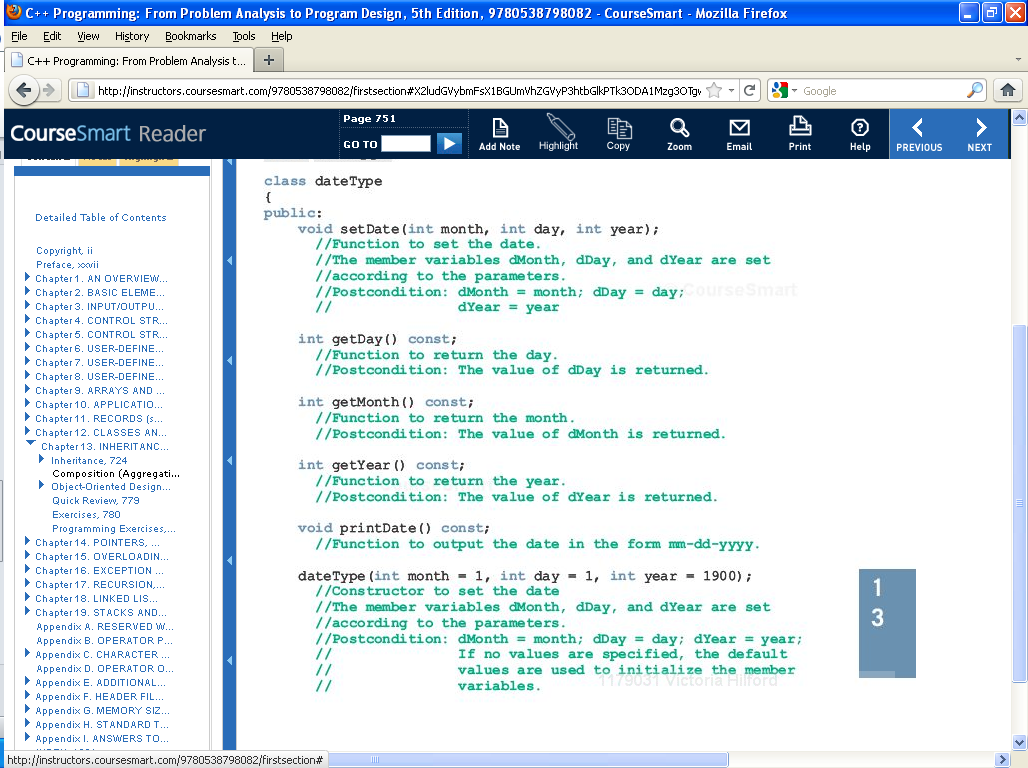
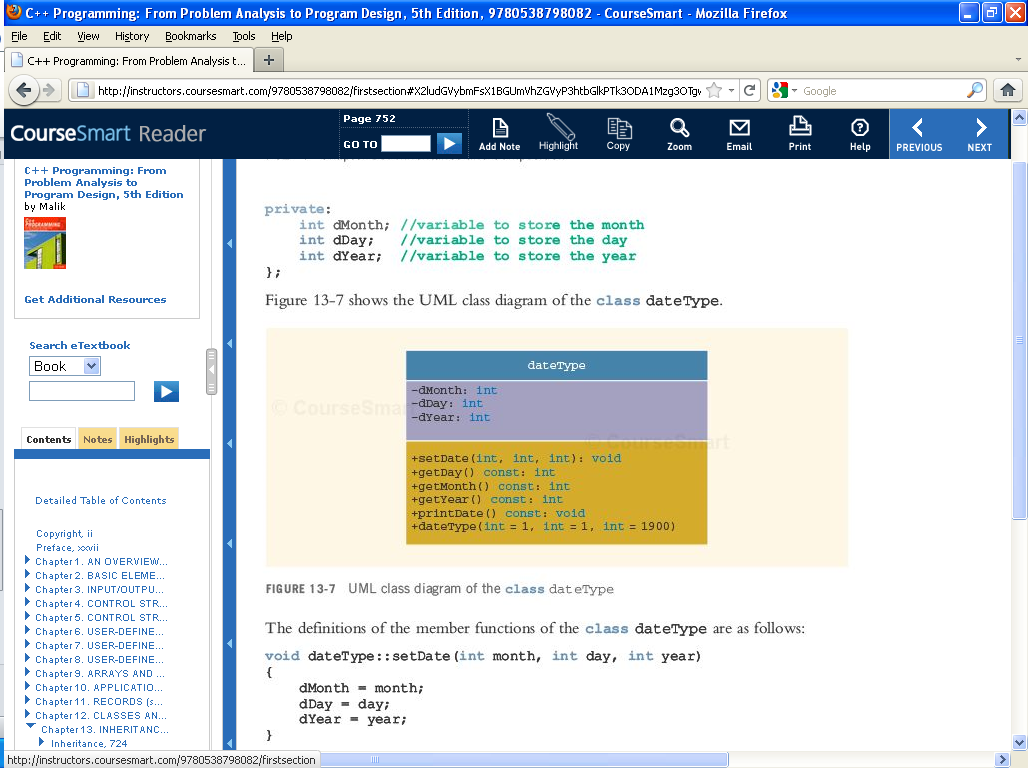
Guess as many parameters as possible.

**ANSWER:**

|  |
| --- |
| ClockType |
| -hr:int //1  -min:int //2  -sec:int //3 |
| +setTime(int): void //1  +getTime():int //2  +printTime(): void //3  +incrementSeconds(): void //4  +incrementMinutes(): void //5  +incrementHours(): void //6  +equalTime(ClockType): void //7 |

**2.** (10 pts) **UML Class Diagram** (**MICROSOFT WORD; Textual Analysis – TA Cut&Paste&Rearrange**). **dateType**

The following statements define the class **dateType** in C++:

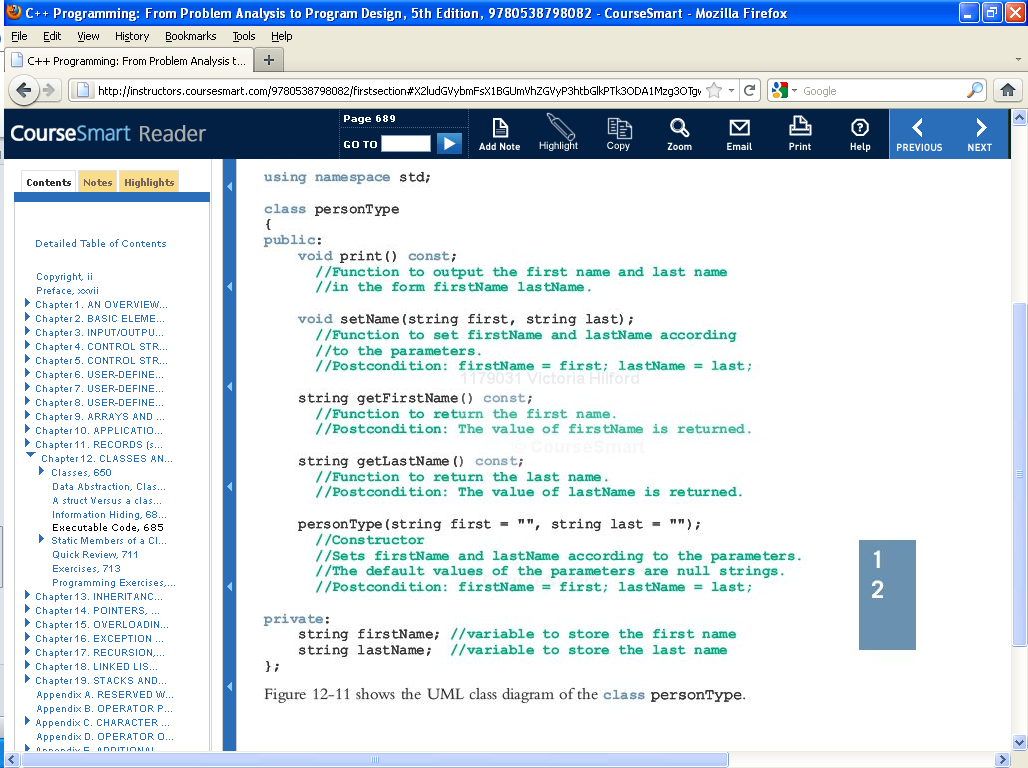
**Draw the UML “compilable” Class Diagram.**

**ANSWER:**

|  |
| --- |
| dateType |
| -dDay:int //1  -dMonth:int //2  -dYear:int //3 |
| +setDate(int,int,int): void //1  +getDay():int //2  +getMonth ():int//3  +getYear (): int //4  +dateType(int, int, int):void //5  +printDate (): void //6 |

**3.** (10 pts) **UML Class Diagram** (**MICROSOFT WORD; Textual Analysis – TA Cut&Paste&Rearrange**). **personType**

The following statements define the class **personType** in C++:



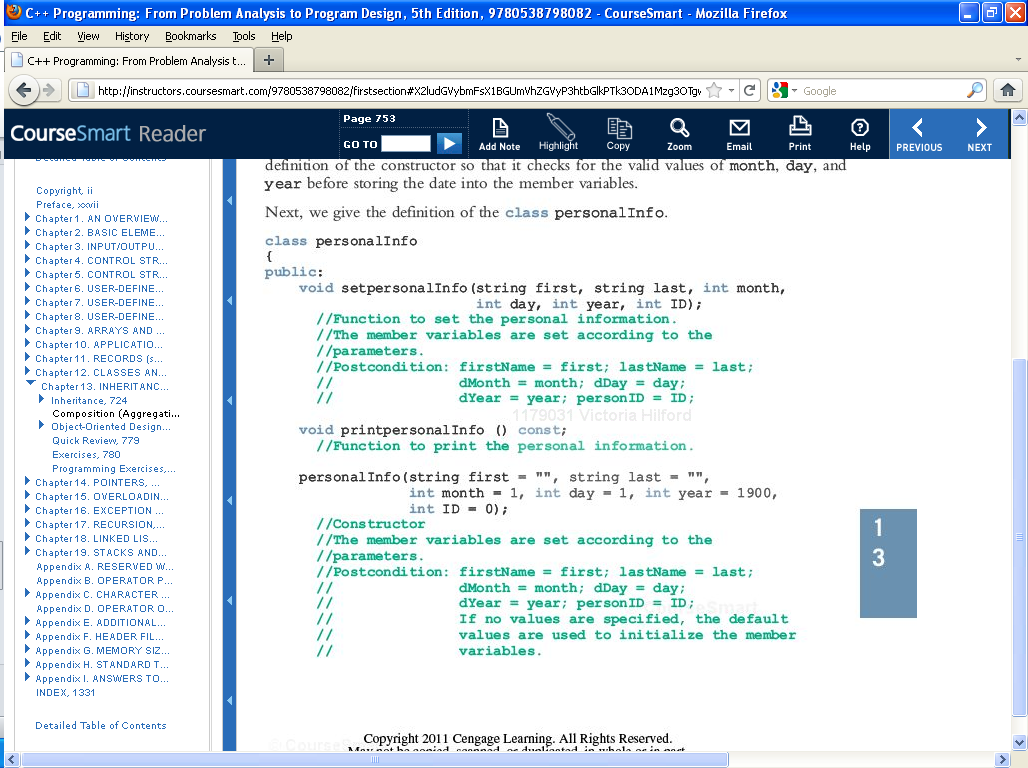
**Draw the UML “compilable” Class Diagram.**

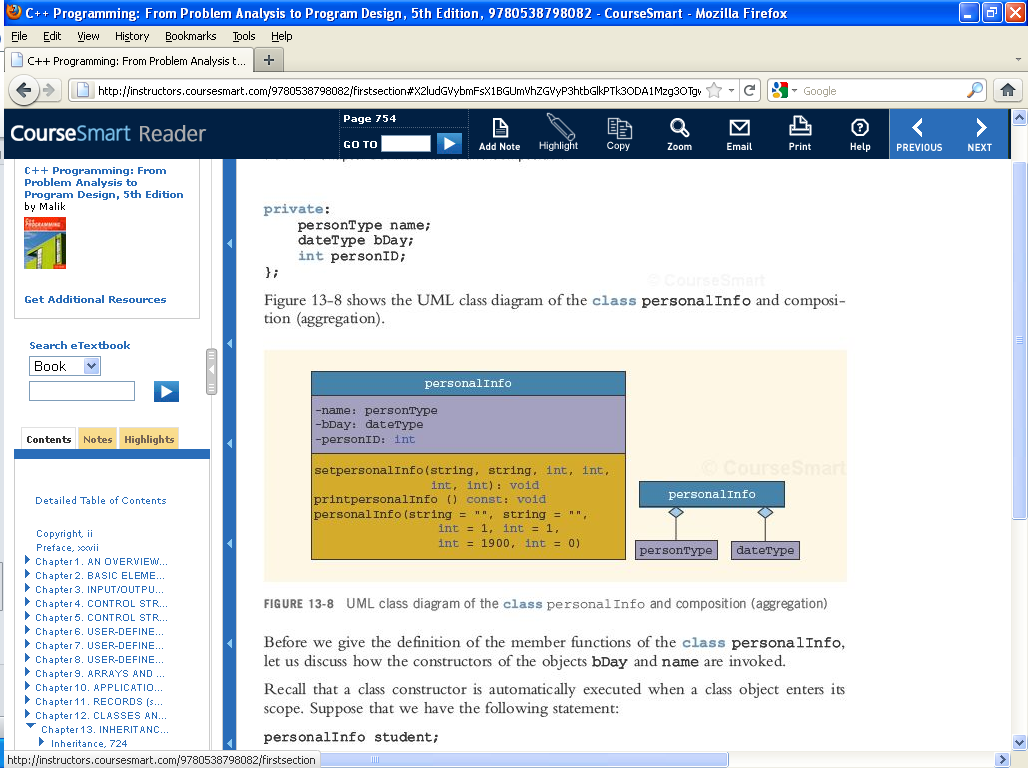
**ANSWER:**

|  |
| --- |
| personType |
| -lastName:String //1  -firstName:String //2 |
| +print (): void //1  +setName (String, String):void //2  +getFirstName ():int//3  +getLastName (): int //4  +personType(String, String):void //5 |

**4.** (10 pts) **UML Class Diagram** (**MICROSOFT WORD; Textual Analysis – TA Cut&Paste&Rearrange**). **personalInfo**

The following statements define the class **personalInfo** in C++:





**Draw the UML “compilable” Class Diagram.**

Pay attention to the relationships between the 3 classes.

**ANSWER:**

|  |
| --- |
| personalInfo |
| -name: personType //1  -bDay: dateType //2  -personID:int //3 |
| +setpersonalInfo(String,String,int,int,int,int):void //1  +printpersonalInfo(): void //2 |

**5.** (30 pts)

**Object Oriented Concepts, UML, and Software Engineering**

**a. (3 pts)**

*A property of design that exploits language mechanisms and encapsulation techniques to prevent client access to, and use of, details about the representation of data structures and types.*

**Answer:**

**b. (3 pts)**

*A programming concept exhibited when two or more* ***methods*** *of the same class have the same name, but different parameter profiles that uniquely distinguish them.*

**Answer:**

**c. (3 pts)**

*The re-definition of a* ***method*** *in a subclass having the same name and equivalent profile as a* ***method*** *in the superclass.*

**Answer:**

**d. (3 pts)**

*A separately named and addressable software unit that encapsulates and abstracts a set of related software capabilities and features. It generally consists of two parts: an interface – for access by similar software units, and an implementation – that specifies how the capabilities and features of the interface are realized.*

**Answer:**

**e. (3 pts)**

*A collection of objects offering a common set of services and possessing a common set of data attributes. It defines or characterizes a new “data type” in the problem or solution domain.*

**Answer:**

**f. (3 pts)**

*A sequence of actions a proposed system performs to offer some results of value to a User.*

**Answer:**

**g. (3 pts)**

*A type of UML diagram that illustrates the interactions between use cases of a given system and the actors that interact with them***.**

**Answer:**

**h. (3 pts)**

*A type of UML diagram used to express the static structure of an OO software design in terms of its modular components and their relationships.*

**Answer:**

**i. (3 pts)**

*A type of UML diagram used to express the dynamic interactions among objects and actors participating in a use case.*

**Answer:**

**j. (3 pts)**

*A type of UML diagram used to express the control flow among processes and use cases at the system level, or the flow of control of algorithmic steps in a method or procedure at the detailed design level.*

**Answer:**

**6.** (10 pts) **UML Class Diagram** (**MICROSOFT WORD; Textual Analysis – TA Cut&Paste&Rearrange**).

Draw a UML Class Diagram for a Class whose objects represent circles. Use Display 20.6 (A UML Class Diagram) as a model.

**ANSWER:**

**7.** (10 pts) **UML Class Diagram** (**MICROSOFT WORD; Textual Analysis – TA Cut&Paste&Rearrange**).

Draw a UML Class Diagram for the IntNode Class presented in Display 17.4 (Functions for Adding a Node to a Linked based **LIST**).

**ANSWER:**

**8.** (10 pts) **UML Class Diagram** (**MICROSOFT WORD; Textual Analysis – TA Cut&Paste&Rearrange**).

One picture is worth a thousand words. Chinese proverb

People do not think in C++ or in any other programming language. As a result, computer scientists have always sought to produce more human- oriented ways of representing programs. One widely used representation is pseudocode, which is a mixture of a programming language such as C++ and a natural language such as English. To think about a programming problem without needing to worry about the syntax details of a language such as C++, you can simply relax the syntax rules and write in pseudocode. Pseudocode has become a standard tool used by programmers, but it is a linear and algebraic representation of programming. Computer scientists have long sought to give software design a graphical representation. To this end, a number of graphical representation systems for program design have been proposed, used, and ultimately found to be wanting. Terms such as flowchart, structure diagram, and many other names of graphical program representations are today only recognized by those of the older generation. Today’s candidate for a graphical representation formalism is the Unified Modeling Language (UML). The UML was designed to reflect and be used with the Object Oriented Paradigm philosophy. It is too early to say whether or not the UML will stand the test of time, but it is off to a good start. A number of companies have adopted the UML formalism for use in their software design projects. History of UML developed along with Object Oriented Paradigm. As the OOP philosophy became more and more commonly used, different groups developed their own graphical or other representations for OOP design. In 1996 Grady Booch, Ivar Jacobson, and James Rumbaugh released an early version of UML. The UML was intended to bring together the various different graphical representation methods to produce a standardized graphical representation language for Object Oriented Design and documentation. Since that time the UML has been developed and revised in response to feedback from the OOP community. Today the UML standard is maintained and certified by the Object Management Group (OMG), a nonprofit organization that promotes the use of Object Oriented techniques. UML Class Diagrams Classes are central to OOP, and the UML Class Diagram is the easiest of the UML graphical representations to understand and use. Display 20.6 shows the UML Class Diagram for a Class to represent a square. The diagram consists of a box divided into three sections. The top section has the Class Name, Square. The next section has the data specification for the Class.

In this example there are two pieces of data (two member variables): a value of type double giving the length of a side, and a value topRtCorner giving the location of the top- right corner of the square.

(The value topRtCorner is given as a pair of numbers of type double, which specify a point in x, y- coordinates relative to some origin.) A minus sign indicates a private member. So, for the Class Square, all data is private. The third section gives the actions (class member functions). A plus sign indicates a public member. A sharp sign, #, indicates a protected member. So for the Class Square, the UML Class Diagram shows two public member functions and one protected member function. A UML Class Diagram need not give a complete description of the Class. When you do not need all the members in a Class for the analysis at hand, you do not list all the members in the UML Class Diagram. Missing members are indicated with an ellipsis (three dots).

Draw a UML Class Diagram for the Square Class presented in Display 20.6.

**ANSWER:**