1. The separation of a data type's logical properties from its implementation.
2. Data encapsulation is the separation of the representation of data from the applications that use the data at a logical level

   A programming language feature that enforces information hiding. When data abstraction is protected through encapsulation, the data user can deal with the data abstraction but cannot access its implementation, which is encapsulated. The data user accesses data that is encapsulated through a set of operations specified to create, access, and change the data. Data encapsulation is accomplished through a programming language feature.
3. Application (or user) level, Logical (or abstract) level and implementation level.

   Application level examples of "a list of student academic records," would be academic institution the students are enrolled or the honor roll students grades

   Logical level examples could be grades/transcripts, student schedules, exams, papers and theses.

   Implementation level examples could be how are the students graded, how do you view the academic records, how are exams, papers, and theses entered.
4. a)Applications of type Grocery Store include theDovv at Bastos, the Super Marche Mahima at Camair, and the Niki at Mokolo.

   b)User operations include SelectItem, CheckOut, PayBill, and so on.

   c)CheckOut operation at logical level:

   <u>CheckOutwith(Basket and Bill)</u>

   Function: Present basket of groceries to cashier to checkout; receive bill.
   Precondition: Basket is not empty.
   Postconditions: Bill = total charge for all the groceries in Basket.
   Basket contains all groceries arranged in papersacks.

   d)CheckOut operation at implementation level:

   <u>CheckOut</u>

```
InitRegister
Set bill to 0
do
OpenSack
while More objects in Basket AND NOT SackFull
Take object from Basket
Set Bill to Bill + cost of this object

Put object in sack
Put full Sack aside
while more objects in Basket
Put full sacks into Basket
```

e)The customer does not need to know the procedure that the Grocery Store uses to check out a basket of groceries and to create a bill. The logical level (c) provides the correct interface, allowing the customer to check out without knowing the implementation of the process.

5. Array, struct, union, and classes.
6. Component selectors for struct, union and class are (.) dot and (->) arrow operators. Dot is used for direct access of members of a struct object. Arrow operator is used for accessing a members of a struct pointer.
7. Public
8. The syntax of the component selector is the array name followed by the index of the desired item: array-name[index-expression].
9. Each array has a base address. The index and the base address are used to access the items in the structure.
10. a)char name[20]

    b)name[0] address location is 1000 so name[9] address location is 1009.

    ..

11.

a) typedef WeatherType WeatherListType[12];

WeatherListType yearlyWeather;

b) yearlyWeather[6].actualRain = 1.05;

c) 238

12

13A) C++ array is used to store the data in the form of a table of rows and columns. ... In C++, a 3d array is a multidimensional array used to store 3-dimensional information. In simple words, a three-dimensional array is an array of arrays. In three dimensional array, we have three rows and three columns.

B) Obviously, this is very useful for data about things in the real world: temperature of each cubic inch of water in a tub, position of each body in the solar system, etc.But it could be ANYTHING describable with 3 indexes. For example, you could have a list of 20 people, each with 10 fingers, each with 3 digits, in your "wart tracker" app!

14 Member-length-offset table for StudentRecord.

Field	*Length*	*Offset*
firstName	10	0
lastName	10	10
id	1	20
gpa	2	21
currentHours	1	23
totalHours	1	24

15. , int x = `1

16.102

17. (a) A square two-dimensional array

(b) struct or class

(c) struct or class containing two data members: the number of quotations and an array of strings that holds them

(d)a one-dimensional integer array

(e) a two-dimensional integer array

(f) a three-dimensional integer array

(g) an array of structs or classes

(h) a one-dimensional integer array

18 .An abstract data type (or ADT) is a class that has a defined set of operations and values. In other words, you can create the starter motor as an entire abstract data type, protecting all of the inner code from the user.

19. In a struct the default visibility is public while in a class the default visibility is private.

20. The members of a class are private unless specified as public. Client code cannot access private members.

21.

22. C++ has something called the One Definition Rule. It means that (excluding inline functions), definitions can only appear in one compilation unit. Since C++ header files are just "copy and pasted" at every include file, now you are putting definitions in multiple places if you just put the definitions in header files.

23. Classes can relate to one another through inheritance, containment (composition), and not at all.

24. Composition is usually used for wrapping classes and to express relationships between classes that contain one another. Inheritance is used for polymorphism, where you have a base class and you want to extend or change its functionality.

25. A base class is a class, in an object-oriented programming language, from which other classes are derived. It facilitates the creation of other classes that can reuse the code implicitly inherited from the base class (except constructors and destructors). A programmer can extend base class functionality by adding or overriding members relevant to the derived class.

A base class may also be called parent class or superclass.

Base classes are automatically instantiated before derived classes.

The derived class can communicate to the base class during instantiation by calling the base class constructor with a matching parameter list.

Base class members can be accessed from the derived class through an explicit cast.

26. No; a derived class does not have access to the private data members of the base class

27. All classes have access to public member functions of any class.

28. a) Specification for an ADT SquareMatrix:

Structure:An N N square integer matrix.
Operations:
MakeEmpty(int n)
Function: Initializes the size of the matrix to n and sets the values to zero.
Precondition: n is less than or equal to 50.
Postcondition: Matrix contains all zero values.

StoreValue(inti, int j, int value)
Function: Stores value into the ith, j th position in the matrix.
Preconditions: Matrix has been initialized; i and j are

between 0 and the size minus 1.
Postcondition: value has been stored into the ith, j th positionof the matrix.

Add(SquareMatrixType one, SquareMatrixType two,SquareMatrixType result)
Function: Adds matrix one and matrix two and stores
the result in result.
Precondition: one and two have been initialized and are thesame size.
Postcondition: result = one + two.

Subtract(SquareMatrixType one, SquareMatrixType two, SquareMatrixType result)
Function: Subtracts two from one and stores the resultin result.
Precondition: one and two have been initialized and are thesame size.
Postcondition: result = one - two.

Print(SquareMatrixType one)
Function: Prints the matrix on the screen.
Precondition: Matrix has been initialized.
Postcondition: The values in the matrix have been printed byrow on the screen.

Copy(SquareMatrixType one, SquareMatrixType two)
Function: Copies two into one.
Precondition: two has been initialized.
Postcondition: one = two.

b)The following declaration contains only additional preconditions and postconditions required by the implementation.

```
classSquareMatrixType
{

public:
void MakeEmpty(int n);
// Pre: n is less than or equal to 50.

// Post: n has been stored into size.
voidStoreValue(inti, int j, int value);
// Pre: i and j are less than or equal to size - 1.

// Post: matrix[i][j] = value
void Add(SquareMatrixType two, SquareMatrixType result);
// Post: result = self + two.
void Subtract(SquareMatrixType two, SquareMatrixType result);
// Post: result = self - two.
void Print();
// Post: The values in self have been printed by row on

// the screen.
void Copy(SquareMatrixType two);
// Post: self = two
private:
int size; // Dimension of the matrix.
int matrix[50][50];
};
```

c)
```
void SquareMatrixType::MakeEmpty(int n)

{
size = n;

for (int row = 0; row < size; row++)
for (int col = 0; col < size; col++)
matrix[row][col] = 0;
}
void SquareMatrixType::StoreValue(inti, int j, int value)
{
matrix[i][i] = value;
}

void SquareMatrixType::Add(SquareMatrixType two,
SquareMatrixType result)
{
int row, col;
for (row = 0; row < size; row++)
for (col = 0; col < size; col++)
result.matrix[row][col] = matrix[row][col] +
two.matrix[row][col];
}
void SquareMatrixType::Subtract(SquareMatrixType two,
SquareMatrixType result)
{
int row, col;
for (row = 0; row < size; row++)
for (col = 0; col < size; col++)
result.matrix[row][col] = matrix[row][col] -
two.matrix[row][col];
}

void SquareMatrixType::Print()
{
int row, col;
for (row = 0; row < size; row++)
for (col = 0; col < size; col++)
cout<< matrix[row][col];
}
void SquareMatrixType::Copy(SquareMatrixType two)
{
int row, col;

for (row = 0; row < size; row++)
for (col = 0; col < size; col++)
matrix[row][col] = two.matrix[row][col];
}
```

d) This test plan is a black-box strategy with data values representing the end cases and a general case. For matrix operations, the end cases represent the size of the matrices,not the values stored in them. We assume that integer addition and subtraction are correct.

Operation to Be Tested and Description of Action	Input Values	Expected Output
MakeEmpty		
execute and print		

29. a)`RelationTypeStrType::ComparedTo(StrTypeotherString)`

```
{
int result;
result = strcmp(letters, otherString.letters);
if (result < 0)
return LESS;
else if (result > 0)
return GREATER;
else return EQUAL;
}
```

b)`RelationTypeStrType::ComparedTo(StrTypeotherString) const`

```
{
int count = 0;

bool equal = true;
while (equal && letters[count] != '\0')
if (letters[count] != otherString.letters[count])
equal = false;
else
count++;
if (otherString.letters[count] == '\0' && equal)
return EQUAL;
else if (equal) // More characters in otherString
return LESS;
else if (letters[count] <otherString.letters[count])
return LESS;
else return GREATER;
}
```

30.
```
void StrType::CopyString(StrType&newString)
 {
 int count = 0;
do
 {
newString.letters[count] = letters[count];

count++;
  }

 while (letters[count-1] != '\0');
 }
```

31. If functions GetMonthAsString and Adjust are called frequently, then the first version is more efficient because access to the string or number of days would be O(1). On the other hand, if these functions are called only once, it would be better to calculate these values and store them. The second alternative would require two new class variables, one to hold the string and one to hold the number of days, but the auxiliary arrays would no longer be need

32.