**CS 37 Notes Part 3**

The next programs are used to illustrate object oriented programming (OOP) which was added to the C programming language to become C++.

|  |  |
| --- | --- |
| structs | Objects |
| When programming with structs, only data members are put into the struct. If I supplied a struct declaration for you to use, it would not be a lot of help to you as you still need to implement all the functionality for the struct. | However, when using objects, the object will contain both the data members and methods (functions within the object) for the object. If I supplied an object for you to use, all you would now need to write is the main since methods have been supplied. **Objects are used for reusable code**, similar to header files. OOP is RAD (rapid application development) for faster software development. Typically objects are put into header files for reusability. |

We will view OOP programming in C++ as using two programmers:

1. the object programmer and
2. the main programmer that includes the class in their programs.

*For our class, you can put the class into a header file and have a .cpp file for the main, or you can put the object and main into one .cpp file.* The sample programs supplied (except for one with a header file) put the objects and mains into one .cpp file. I do suggest that you write at least one program that does use a header file for your object for practice.

Vocabulary (OOP terms):

Encapsulation (within) – Data members and methods are encapsulated (put within) the objects.

Class – the programming code for an object. We will be replacing the word struct in C with class in C++. For example, struct Student will become class Student in our C++ programs.

Data Access Specifiers specify the extent of access the data members and methods can be accessed. There are 3 possibilities:

1. Public – the data member and method can be accessed anywhere in the program.
2. Private – the data member and method can only be accessed within the class methods only.
3. Protected – the data member and method can only be accessed in other classes that *inherit from the class*. We will cover the protected data access specifier later when we cover inheritance.

eg public:

int x, y, z; x, y, z, can be accessed throughout the program, whether in the main or in the class

private:

float a, b, c; a, b, c, can only be accessed in the class methods. The main programmer does not have access to a, b, c.

**Dot notation is used to access data members and methods in a class.**

Example:

Suppose we have a Student class with data members t1, t2, t3 and a method called calc().

First, an object must be declared or instantiated (create an instance of).

|  |
| --- |
| Student s; |

We can do the following to read in the three data members in the main assuming the 3 data members are public:

|  |
| --- |
| cout << “Enter three scores “;  cin >> s.t1 >> s.t2 >> s.t3; (same as we did for structs in C) |

To call a method, you can do the following in the main assuming calc() is also public:

|  |
| --- |
| s.calc(); |

Because student s; and calc();

If they were private instead, the code above will give you errors because you can’t access t1, t2, t3, or calc() from the main

**class1.cpp**

demonstrates public vs private

*This program has a class Student which only contains data members. The class begins with the code class Student followed by the declarations of the data members name, test1, test2, test3, and avg that are all public. Note the keyword public: above the data members. If you leave off the data access specifier, the data members (and methods later) will be considered as private. In the main, an object s is instantiated, the values are entered by the user, the average is calculated and the results are printed. Note that the class ends with }; Like structs in C, semicolons are needed at the end of each class.*

**class2.cpp**

The main difference between program 2 and program 3 is that program 2 has all the functions stored within the program while program 3 accesses student.h and then run the functions from there

|  |
| --- |
| #include<iostream>  using **namespace** std;  *// adding some methods (functions for the objects) to the class*  **class** Student{ This is how you “initiate” the constructor student  **public:**      Student();*default constructor has a method which is public and therefore the constructor is public*  *// constructor is automatically called when an object is instantiated*  *// constructor MUST have the same name as the class*  **void** setData(**int**, **int**, **int**);  **void** printData();  **private:**  **int** test1, test2, test3;  **float** avg;  };  Student::Student()      { test1 = test2 = test3 = 1;        avg = (test1 + test2 + test3) / (**float**)3; }  **void** Student::setData(**int** a, **int** b, **int** c)  {      test1 = a;      test2 = b;      test3 = c;      avg = (test1 + test2 + test3) / (**float**)3;  }  **void** Student::printData()  {      cout << "test 1 is " << test1 << endl;      cout << "test 2 is " << test2 << endl;      cout << "test 3 is " << test3 << endl;      cout << "the average is " << avg << endl;  }    **int** main()  {      Student one, two;*// constructor is called once for each object instantiated*      cout << "student one data is" << endl;      one.printData();      two.setData(10, 15, 30);      cout << endl << "student two data is"  <<endl;      two.printData();      return 0;  }  */\**  *student one data is*  *test 1 is 1*  *test 2 is 1*  *test 3 is 1*  *the average is 1*  *student two data is*  *test 1 is 10*  *test 2 is 15*  *test 3 is 30*  *the average is 18.3333*  *Press any key to continue . . .*  *\*/* |

This program adds three methods to our class:

one.printData();

two.setData(10, 15, 30);

two.printData();

*The class in this program begins with the public declarations for the three methods.* Again note that the constructor must be called Student() for the Student class.

Student one, two;

The private section is used to declare the 3 test scores and the average data members. Note that these 4 data members cannot be accessed directly from the main as they are private. We are now outside the scope of the class, past the };

We will need the scope resolution operator :: to tell C++ that the three methods are for the Student class. You can certainly have more than one class in a program. Each method would need the class name that it corresponds to. For example, void Student :: setData(int a, int b, int c) specifies that setData is associated with the Student class.

The constructor method is implemented first and sets all the data members of the class to 1.The setData method is next and it copies over the parameter values to the data members of the class and calculates the average data member (avg). The printData method prints out the values of the data members for the object.

The main instantiates two objects. Initially the values are all 1’s for both objects due to the constructor. The first object one prints out with all 1’s. The setData method is used to assign the values for object two and the printData method is then called for object two.

Because the data members are private in this program, the main programmer has no direct access to these data members. The main programmer can only use the given methods to work with the data members. This gives the class programmer the ability to limit access to data members from the main. For example, if gross pay was to be calculated, then gross pay should be private as we do not want the main programmer to be able to the following in the main:

gross = 99999.99;

1. Constructor: automatically called when an object is instantiated.
   1. Why? To initialize object values and to handle the code that needs to be done once. For example, set sums and counter to 0 or print initial directions for the user.
   2. Has the same name as the class

**Constructor 1:** Student(), when we instantiate objects such as:

|  |
| --- |
| Student a, b; |
| Process for what happens when student a, b; is instantiated:   1. Allocate memory for Student a 2. Constructor is automatically called on student a 3. Memory is allocated for Student b 4. Constructor is automatically called on Student b |

**Constructor 2:** setData method

* Has 3 parameters which are passed
* 3 values are placed into the data members of the class

**Constructor 3:** printData

* prints the values of the object

*On method names that have more than one word, it is accepted style to capitalize the first letter of each word except for the first word. eg setData , printData, searchTheData.*

It is also common to use error checking in the set methods such as setData. A series of if statements can be used to make sure a valid value is entered into each data member. For example, no negative gross pays or no test scores outside the range of 0 to 100. You could use a loop in the set method which asks the user to enter in a value for a data member until a valid data member value is entered, eg 0 to 100.

**class3.cpp**

This program is identical to class2.cpp but uses a header file called student.h that contains the class code. The program begins with the usual include and using namespace std statements; but also includes the line #include”student.h”

#include "student.h"



For include statements, system header files such as iostream go in <> without the .h while user header files go in double quotes and need the .h.

Comparison to the previous program:

* Main is the same
* The header file student.h contains the class code from the previous program only, no main.
* \**Note that the iostream must be included into the header file along with using namespace std.*

To enter this program, you first add a new item for the student.h header file and type in the class code. Next, you add another item for the class3.cpp file and enter the main.

You will see two tabs at the top of the screen, one for the header file and one for the .cpp file that are used to move between the two files. Just run the program in the usual way.

#include "student.h"

**class4.cpp**

|  |
| --- |
| #include<iostream>  #include<iomanip>  using **namespace** std;    **class** Person  {  **public:**      Person();*// constructor*  **void** getSales();*// get sales amount from keyboard*  **void** printSales();*// print the sales array*  **private:**  **float** totalSales();*// private utility function*  **float** sales[5];*// data members can be arrays*  };  Person::Person()*// constructor function sets array elements to 0.0*  {      for (**int** i = 0; i < 5; i++)          sales[i] = 0.0;  }  **void** Person::getSales()  {      for (**int** i = 1; i < 6; i++)      {          cout << "enter amount for month " << i << "   ";          cin >> sales[i-1];      }  }  **void** Person::printSales()  {      cout << endl;      cout << setprecision(2) << showpoint << fixed;      for (**int** i = 0; i < 5; i++)          cout << sales[i] << "  ";          cout <<endl << endl;      cout << "total sales are " << totalSales() << endl;  }  **float** Person::totalSales()  {  **float** sum = 0.0;      for (**int** i = 0; i < 5; i++)          sum += sales[i];      return sum;  }  **int** main()  {      Person p;      p.getSales();      p.printSales();      return 0;  }  */\**  *enter amount for month 1   10.22*  *enter amount for month 2   20.33*  *enter amount for month 3   30.44*  *enter amount for month 4   40.55*  *enter amount for month 5   50.66*  *10.22  20.33  30.44  40.55  50.66*  *total sales are 152.20*  *Press any key to continue . . . \*/* |

\*doesn’t include a special header file like the last one

Main:

* p.getSales();
  + Inputs sales amount for each month into an array
* p.printSales();
  + prints out the sales array
  + calls totalSales(); and prints it
    - returns total

This program uses an array as a data member

**private:**

**float** sales[5];*// data members can be arrays*

And also demonstrates a private utility function that can only be called from a class method.

**class** Person

{

**public:**

    Person();*// constructor*

**void** getSales();*// get sales amount from keyboard*

**void** printSales();*// print the sales array*

**private:**

**float** totalSales();*// private utility function*

**float** sales[5];*// data members can be arrays*

};

Called inside:

**void** Person::printSales()

{

    . . .

    cout << "total sales are " << totalSales() << endl;

}

Once again, the class programmer is limiting what the main programmer has access to.

Demonstrates that C++ does allow an array of objects to be “declared *outside of main”*. Suppose we had 10 sales people, the main could be rewritten as follows. *Each time through the for loop, one sales person is processed.*

**int** main()

{

Person p[10];

for (**int** i = 0; i < 10; i++)

{

        p[i].getSales();

        p[i].printSales();

}

return 0;

}

Overview

The class Person begins with public declarations for the constructor, getSales method, and printSales method.

**public:**

    Person();*// constructor*

**void** getSales();*// get sales amount from keyboard*

**void** printSales();*// print the sales array*

The private section contains declarations for a private utility function totalSales (I do not like that these are called private utility functions, if it was up to me, the term would be private utility methods.), and a 5 float array called sales. The array will keep track of monthly sales of an item for 5 months.

**private:**

**float** totalSales();*// private utility function*

**float** sales[5];*// data members can be arrays*

The constructor code initializes the array to elements to 0.0 using a for loop.

Person::Person()*// constructor function sets array elements to 0.0*

{

    for (**int** i = 0; i < 5; i++)

        sales[i] = 0.0;

}

The getSales method allows the user to enter the total sales for months 1 through 5. Note the loop goes from 1 to 5 to correspond to the months but the cin statement uses i – 1 as the array elements go from 0 to 4.

**void** Person::getSales()

{

    for (**int** i = 1; i < 6; i++)

    {

        cout << "enter amount for month " << i << "   ";

        cin >> sales[i-1];

    }

}

The printSales method prints out the array and calls the totalSales method so that the total sales is returned and printed.

**void** Person::printSales()

{

    cout << endl;

    cout << setprecision(2) << showpoint << fixed;

    for (**int** i = 0; i < 5; i++)

        cout << sales[i] << "  "; // prints out

        cout <<endl << endl;

    cout << "total sales are " << totalSales() << endl; // calls totalSales()

}

The totalSales method uses a for loop to sum up the array and return the sum.

**float** Person::totalSales()

{

**float** sum = 0.0;

    for (**int** i = 0; i < 5; i++)

        sum += sales[i];

    return sum;

}

The main instantiates a Person p and calls the getSales and printSales methods.

**int** main()

{

Person p;

p.getSales();

p.printSales();

return 0;

}

**class5.cpp**

This program adds a destructor method to the Student class. This program is the same as class2.cpp but the destructor has been added.

**class** Student{

**public:**

    Student();*//default constructor*

    ~Student();*// destructor*

*What is a destructor?*

The destructor method is automatically called when an object goes out of scope. The destructor must have the same name as the class with a tilde in the front. For example, ~Student(). (The tilde ~ is at the top left of the keyboard and is shift `)

*For example:*

* Local variables declared in functions can only be accessed in the function they’re declared BECAUSE At the end of a function, all the local variables are automatically destroyed.
* Objects are similar. When an object goes out of scope, the destructor method is called automatically, and the object gets destroyed!

Destructor code contains the code that needs to be done once at the end of a program*.* For example, to print out sums and counters or in a video game when the character has died in the game.

The destructor is declared in the public section

**public:**

    Student();*//default constructor*

    ~Student();*// destructor*

and the destructor simply prints that a student has been destructed along with their first test score.

Student::~Student()

{

    cout << "a student has been destructed" <<endl;

    cout << "first test score for the student is " << test1 <<endl;

}

This program is meant to demonstrate the destructor method only as you would not have a destructor with this functionality in your programs. *Later on, destructors will be used to return the object to free memory so that the memory can be reused by other objects later in the program.*

*If you are using Visual Studio, you will have to use the system(“PAUSE”); command at the end of the destructor. The destructor will be automatically called after the return 0; in the main is executed. If there is no system(“PAUSE”); in the destructor, then you will not see any destructor output.*

Please note the output of this program as the destructor calls are made in reverse order of the constructor class. This program instantiates Student one first and then Student two. However, the destructor for Student two is called first followed by the destructor for Student one.

student one data is

test 1 is 1

test 2 is 1

test 3 is 1

student two data is

test 1 is 10

test 2 is 20

test 3 is 30

a student has been destructed

first test score for the student is 10

a student has been destructed

first test score for the student is 1

You may have heard about a stack which is a LIFO structure for last in first out. Objects are put onto a stack as they are instantiated. In this program, object one is put on the stack first and then object two is put on the top of object one (think of cafeteria trays, the last tray put on the stack is the first tray removed from the stack). When the objects go out of scope, object two at the top of the stack is destroyed first followed by object one being destroyed.

|  |
| --- |
| #include<iostream>  using **namespace** std;  **class** Student{  **public:**      Student();*//default constructor*      ~Student();*// destructor*  *// destructor is automatically called when an object goes out of scope*  *// destructor MUST have the same name as the class with a tilde ~ in front*  **void** setData(**int**, **int**, **int**);  **void** printData();    **private:**  **int** test1, test2, test3;  };  Student::Student()      { test1 = test2 = test3 = 1;}  Student::~Student()  {      cout << "a student has been destructed" <<endl;      cout << "first test score for the student is " << test1 <<endl;  }  **void** Student::setData(**int** a, **int** b, **int** c)  {      test1 = a;      test2 = b;      test3 = c;  }  **void** Student::printData()  {      cout << "test 1 is " << test1 << endl;      cout << "test 2 is " << test2 << endl;      cout << "test 3 is " << test3 << endl;  }  **int** main()  {      Student one, two;      cout << "student one data is" << endl;      one.printData();      two.setData(10, 20, 30);      cout << endl << "student two data is"  <<endl;      two.printData();      cout << endl <<endl;      return 0;  }  */\**  *student one data is*  *test 1 is 1*  *test 2 is 1*  *test 3 is 1*  *student two data is*  *test 1 is 10*  *test 2 is 20*  *test 3 is 30*  *a student has been destructed*  *first test score for the student is 10*  *a student has been destructed*  *first test score for the student is 1*  *Press any key to continue*  *Note destructor calls are made in the reverse order of the constructor calls.*  *\*/* |

**class6.cpp**

C++ visualization:

PART 1:

* call student(); student calls setdata(); and setdata(); defaults to 0 0 0 for one
* repeats process for two and three and four except they put diff numbers in because the numbers put in diff parameters

**int** main()

{

    Student one, two(50), three(50,70), four (50,70,90);

PART 2:

* after that’s all done, we go back to main and then print everything out depending on which instance it is, and just goes to printdata(); depending on the parameters of one

    cout << "student one data is" << endl;

    one.printData();

* this just repeats itself inside of main

    cout <<endl << "student two data is"  <<endl;

    two.printData();

    cout <<endl << "student three data is"  <<endl;

    three.printData();

    cout <<endl << "student four data is"  <<endl;

    four.printData();

PART 3:

* The program finishes inside of main

    cout <<endl <<endl;

    return 0;

* Goes back to this line

    Student one, two(50), three(50,70), four (50,70,90);

* Then destructor is executed

Student::~Student()*// destructor - called when object is destroyed*

{

    cout << setw(10) <<test1 << setw(10) << test2;

    cout << setw(10) << test3 << " destroyed" <<endl;

}

* Then the destruction goes down with student four, student three, student two, student one

50 70 90 destroyed

50 70 0 destroyed

50 0 0 destroyed

0 0 0 destroyed

This program adds default arguments to our constructors similar to the default arguments we covered back in the function material. Like default arguments in the past, only the rightmost arguments can be defaulted in the constructor.

The Student object has only three data members for three test scores. The constructor declaration in the public section is:

Student(int = 0, int = 0, int = 0);

By declaring default arguments, we can do the following instantiations in the main:

Student one, two(50), three(50, 70), four(50, 70, 90);

In the main, you can instantiate and specify initial values for the data members. If no values are passed, then the data members are set to the default value of 0. For the instantiations given, the following values are assigned to the test scores of each object:

one: 0 0 0

two: 50 0 0 (only the rightmost arguments are defaulted)

three: 50 70 0

four: 50 70 90

This program also demonstrates how one method can call another method like we did with calling totalSales from printSales in class4.cpp. The constructor method calls the setData method to assign the values to each data member. The destructor once again shows that the destructor calls for the objects are done in reverse order of the constructor calls.

**class7.cpp**

This program has the same class as class6.cpp. The main has been modified to demonstrate membership copy where all data members of one object are assigned to the corresponding data members of the other object.

|  |
| --- |
| Class 6 |
| **int** main()  {      Student one, two(50), three(50,70), four (50,70,90);        cout << "student one data is" << endl;      one.printData();      cout <<endl << "student two data is"  <<endl;      two.printData();        cout <<endl << "student three data is"  <<endl;      three.printData();        cout <<endl << "student four data is"  <<endl;      four.printData();        cout <<endl <<endl;        return 0;  } |
| Class 7 |
| **int** main()  {      Student one, two(50,70), cpone, cptwo;      cout << "student one data is" << endl;      one.printData();  *// can copy objects using default membership copy*      cpone = one;      cout <<endl << "student one's copy data is"  <<endl;      cpone.printData();        cout <<endl << "student two data is"  <<endl;      two.printData();      cptwo = two;      cout <<endl << "student two's copy data is"  <<endl;      cptwo.printData();      cout <<endl <<endl;      return 0;  }  *50        70         0 destroyed*  *0         0         0 destroyed*  *50        70         0 destroyed*  *0         0         0 destroyed* |

For example, if we have two student objects one and two and the following line of code is done:

two = one; All the data members of object one are assigned to the corresponding data members of object two. That is test 1 of object two is set to test 1 of object one, test 2 of object 2 is set to test 2 of object 1, and test 3 of object 2 is set to test 3 of object 1.

    cout << "student one data is" << endl;

    one.printData();

*// can copy objects using default membership copy*

    cpone = one;

    cout <<endl << "student one's copy data is"  <<endl;

    cpone.printData();

After this line of code is executed, both objects will have the same data member values which are the values of object one. (We did membership copy also in C to copy all the data members from one struct to another struct.)

Object one was copied over to cpone and object two was copied over to cptwo

**class8.cpp**

This program uses set functions (mutators) to assign a value to one data member of the object and get functions (accessors) to return a value of one data member of an object. (Again, I do not like these being called functions as they should be called methods, but functions is the accepted terminology to use.) The set and get functions will be used in class9.cpp to modify the value of a private data member.

The class declaration begins with a constructor with default arguments and a destructor for illustration purposes only.

**class** Student{

**public:**

    Student(**int** = 0, **int** = 0, **int** = 0);*//default arguments with constructors*

    ~Student();*// destructor*

Step 1: print one’s data as is

    one.printData();

Step 2: set two’s data inside main

    two.setTest1(10);

    two.setTest2(20);

    two.setTest3(30);

Step 3: Print two’s data

    cout <<endl << "student two data is"  <<endl;

    two.printData();

    cout <<endl;

Step 4: Note that these functions in main are mutators The setData method is declared followed by the three set functions called setTest1, setTest2, and setTest3. Each of the set functions take one parameter which is the value to assign to the respective data member of the object. The setData method calls each of the set functions. The constructors inside student(); simply calls the setData method to assign all three test scores.

inside main

    qz1 = two.getTest1();

    qz2 = two.getTest2();

    qz3 = two.getTest3();

inside student()

**void** setData(**int**, **int**, **int**);

**void** setTest1(**int**);

**void** setTest2(**int**);

**void** setTest3(**int**);

inside the program

**void** Student::setData(**int** a, **int** b, **int** c)

{

    setTest1(a);

    setTest2(b);

    setTest3(c);

}

**void** Student::setTest1(**int** a) { test1 = a;}

**void** Student::setTest2(**int** b) { test2 = b;}

**void** Student::setTest3(**int** c) { test3 = c;}

Step 5: Because of step 4,

    qz1 = two.getTest1();

    qz2 = two.getTest2();

    qz3 = two.getTest3();

You can just do something like this to call the functions:

    cout << "using the get functions" << endl;

    cout << setw(10) << qz1 << setw(10) << qz2;

    cout << setw(10) << qz3 << endl << endl;

The get functions are declared next followed by a printData method and three private data members for the three test scores.

The destructor prints out the object being destroyed when the object goes out of scope.

Each set function assigns the value to the respective data member. For example, setTest1 assigns the parameter passed to the data member test1.

Each get function simply returns the respective data member.

The printData methods prints out the values of the object.

The main instantiates two objects. The first object is printed with the default values of all zeroes. Each set function is called next for object two and object two is then printed. The get functions are each called and the values returned are put in the variables qz1, qz2, and qz3. Lastly the three variables qz1, qz2, and qz3 are printed.

If the function is un-deterministic IE. If you need an input from the user, Usually the set functions will have bounds checking code to make sure that the values to put into the object are valid. For example for setTest1, we could do the following code:

if(a >= 0 && a <= 100)

test1 = a;

else

{

              cout << “ invalid score entered” << endl;

     do {

                     cout << “enter a valid test 1 score  “;

                     cin >> a;

                     if(a < 0 || a > 100)

                             cout << “Invalid score entered”  << endl;

              }  while (a < 0 || a > 100);

         }

If a valid parameter is passed, then the value is put into the data member of the object; otherwise, a loop is used that keeps asking for the user to enter a test score until a valid score is entered for cin >> a.

This program basically shows that you can assign a value to a privately declared variable inside a construct through a method function

**class9.cpp**

This program uses the same class as class8.cpp. The main programmer has added a function called change() to add 10 points to test2 and 20 points to test3.

**void** change( Student **&**);*// prototype for function to change a student's values*

***Change is a function and not a method of the class.*** After the class, the change function is prototyped, note the function takes one reference parameter for the object. And this is kind of weird!

**void** change(Student **&**pupil) *// add extra credit for test 2 and test 3*

{

    pupil.setTest2(pupil.getTest2() + 10);

    pupil.setTest3(pupil.getTest3() + 20);

}

*student two data is*

*test 1, 2, 3 are         10        20        30*

*using the get functions*

*10        20        30*

*after the change function*

*test 1, 2, 3 are         10        30        50*

*10        30        50 destroyed*

As the values need to be passed back, the object is a reference parameter. The main uses the line change(two); to call the change function. Object 2 is printed before and after the change function is called.

    cout << "using the get functions" << endl; // using the get funcs through qzs

    cout << setw(10) << qz1 << setw(10) << qz2; // printing before change begins

    cout << setw(10) << qz3 << endl << endl;

    change(two);

\*Because the data members are private in the class, you cannot do the following line of code in the change function:

pupil.test2 += 10; as test2 cannot be accessed in the function as it is private.

Instead, what is done is that we first get the value of test2, add 10 to it, and then use the set function to put the new value back into the object. That is:

    pupil.setTest2(pupil.getTest2() + 10);

What this line of code does is that it goes back to this method:

**int** Student::getTest2() { return test2; }

Recall that test2 = 20

    two.setTest2(20);

Then because we replace test2 = pupil.getTest2() = 20, 20 + 10 = 30. Therefore, the newTest2 after change(); is 30, replacing the value of test2:

    pupil.setTest2(pupil.getTest2() + 10);

Output:

*student two data is*

*test 1, 2, 3 are         10        20        30*

*using the get functions*

*10        20        30*

*after the change function*

*test 1, 2, 3 are         10        30        50*

Destructor:

*10        30        50 destroyed*

KEY TAKEAWAY:

Basically what this program wants to convey is that you can add new functions like

**void** change(Student **&**pupil)

And change pre-existing values with the functions that are used to get those values

    pupil.setTest2(pupil.getTest2() + 10);

**class10.cpp**

This program uses a method called add to find the sum of the corresponding data members of two objects.

For example, the call could be three.add(one,two);

    three.add(one,two);

For example, suppose one has test scores of 10, 20, 30 and two has test scores of 40, 50, 60. The add function will assign the values 50, 70, 90 to the three test scores of object three. test1 of object three will be assigned the sum of test 1 of objects one and two. Same for the other two tests for object three.

The add function requires two parameters corresponding to the two objects to be added. For test1, the code is test1 = a.test1 + b.test2; In the call three.add(one,two); one is passed to Student a and two is passed to Student b in the parameter list. In the add method, dot notation is used on the right hand side of the code, the test1 on the left side corresponds to object three in the call.

Please note the output as the destructor is called several times in this program. In the add function, Student a and Student b are destroyed at the end of the add function so the destructor is called on both objects. Student a is destroyed first followed by Student b. As mentioned earlier, destructors are called in reverse order of instantiations but parameters are also done in this order. In other words, a is destroyed first and b is destroyed second. At the end of the main, the three objects of the main are destroyed in reverse order. Once again, the destructor is used in this program for illustrative purposes only.