Programming Project 2

For your second programming project, you will explore the use of inheritance, virtual functions, and linked lists. To begin, you can unpack the project2 zip file available on WebCT. Then open the project2 project file and review the various files. You should see that the project contains several files including: project2.h, project2.cpp, list.h, list.cpp, resource.h, and resource.rc that can be used to construct a simple graphical drawing package.

There are two parts to this project. One part involves code for a graphical interface for a simple drawing package and the other part involves the data structures used to store the graphical objects. We have implemented most of the code needed for the graphical interface and placed it in the files: project2.h, project2.cpp, resource.h and resource.rc. The graphical interface creates a menu that allows a user to draw four different types of objects (squares, rectangles, circles, and ellipses) in a variety of colors. You can also select objects once they have been drawn and remove them or change the order in which they displayed. The other part of the program is the list structure that stores descriptions of the graphical objects. This list is critical to the operation of the program because whenever the window containing the drawing objects changes, some or all of the objects must be redrawn. The typical approach to solving this problem is to maintain a linked list of objects so that when it is necessary to redraw the screen, the graphical interface can traverse the list and draw each object in sequence. Since the graphical interface must know the names of the objects, we will define a special class called BaseObject as shown below.

class BaseObject
{
    public:
        BaseObject(const CRect&, const COLORREF&); // constructor
        virtual void draw() = 0; // virtual drawing function
        void bounds(const CRect&); // set the bounding rectangle
        void color(const COLORREF&); // set the color
        CRect bounds(); // retrieve the bounding rectangle
        COLORREF color(); // retrieve the color
        bool ptInside(CPoint& pt); // check if point is inside
    protected:
        CRect boundingRect; // bounding rectangle
        COLORREF objectColor; // color
};

The BaseObject class stores two important pieces of information about each object, the bounding rectangle and the color of the graphical image. The bounding rectangle defines the corner points of a box where the object will be draw. The bounding box is declared as type CRect. This is a class defined in the MFC (Microsoft Foundation Class) library. If you look in the online help manual under the listing for CRect you will find all the operations available on rectangles. Colors are represented as type COLORREF. This is also part of the MFC library. A COLORREF is just an unsigned integer value that describes a color. The format is usually shown in hexadecimal form: 0x00rrggbb where the hexadecimal digits for rr determine the amount of red in the color, gg determines the amount of green and bb determines the amount of bb. For example, a deep red color is 0x00ff0000. The class constructor allows the user to create a BaseObject containing a given bounding rectangle and color. The bounds and color member functions allow the user to retrieve or set the value of the data members. The ptInside function is used to decide if a point (MFC CPoint object) is inside the bounding rectangle. The draw function is declared as a pure virtual function. This function is called by the graphical interface to draw objects. Since the function is pure virtual, there is no implementation provide in the BaseObject class. Instead, the implementation is provided in classes that are derived from the BaseObject class. You will need to implement four additional object classes: Square, Rectangle, Circle, and Ellipse. These classes are responsible for drawing specific shapes. Take a closer look at the Square class on the next page.
class Square : public BaseObject
{
public:
    Square( const CRect&, const COLORREF& ); // constructor
    void draw(); // drawing function
};

This class is derived from a BaseObject so it has all the functionality of a BaseObject plus it provides a constructor for initializing a Square object and a drawing function that is called through a list function (to be described shortly). This is an extremely simple class, and the implementation is provided in list.cpp.

The final two classes you need to implement are: Node and List. The class definitions are shown below.

class Node
{
private:
    Node( BaseObject* obj = NULL, Node* ptr = NULL );
    BaseObject* data;
    Node* next;

    friend class List;
};

The Node class has a constructor that is used to initialize the fields of the Node and two data members. The BaseObject* data field is used to store a pointer to a BaseObject that stores the description of a graphical shape and the Node* next field is used to chain nodes together to form a linked list. The Node is not accessed directly by the interface so you are free to change the class definition.

class List
{
public:
    List();
    List( const List& );
    virtual ~List();
    List& operator= ( const List& );
    void insert( string, const CRect&, const COLORREF& );
    void display();
    BaseObject* find( CPoint );
    void remove( BaseObject* );
    void removeAll();
    void moveToTop( BaseObject* );
    void moveToBottom( BaseObject* );
    void moveUp( BaseObject* );
    void moveDown( BaseObject* );

private:
    // detail of the private section are your choice
};

The List class interacts with the interface so it is important that you do not change any of the functions listed above. You are free to define anything you want in the private section of the class. The purpose of each function is described on the next page.
List()  
-- class constructor

List( const List& )  
-- list copy constructor

virtual ~List()  
-- list destructor

List& operator=( const List& )  
-- overloaded list assignment operator

void insert( string, const CRect&, const COLORREF& )  
-- insert a new description of graphical object in the list. The string will contain one of the following values: “square”, “rectangle”, “circle”, or “ellipse”. This will indicate the type of shape to store. You will not need the string after the object (Square, Rectangle, Circle, Ellipse) is created so there is no need to store it. You will also be passed a CRect object and COLORREF value to store in the object.

void display( )  
-- display the set of stored objects. This function should traverse the list and display all objects by invoking their draw( ) function. It is important that objects are displayed in the order they are created because one object may be drawn on top of another object. Notice the move operations described below may alter this order.

BaseObject* find( CPoint )  
-- find the object whose bounding box contains the point. If the object is found return a pointer to the object. If no object is found, return a NULL value. If more than one object’s bounding box contains the point, display the topmost (most recently created) object.

void remove( BaseObject* )  
-- remove and destroy the data object referenced by the BaseObject pointer. Also destroy the Node containing the data object. If the BaseObject is NULL, take no action.

void removeAll( )  
-- remove and destroy all objects and nodes on the list.

void moveToTop( CBaseObject* )  
-- find the Node containing the BaseObject data pointer and move the Node / BaseObject pointer to a position in the list so it is the last object displayed.

void moveToBottom( CBaseObject* )  
-- find the Node containing the BaseObject data pointer and move the Node / BaseObject pointer to a position in the list so it is the first object displayed.

void moveUp( BaseObject* )  
-- find the Node containing the BaseObject data pointer and move the Node / BaseObject pointer to a position in the list so it is displayed one item later in the list.

void moveDown( BaseObject* )  
-- find the Node containing the BaseObject data pointer and move the Node / BaseObject pointer to a position in the list so it is displayed one item earlier in the list.

Make sure to fully document all list operations. After you have developed your solution, compile and link the project and test all functions and sequences of function calls to make sure you have the correct implementation. On the assigned due date, submit all source files (.h/.cpp) using WebCT.