Programming Assignment 3: VLSI Design Automation, Due Apr 22.

Create an ActiveX control with the following characteristics:

It is invisible at run time.
It has one property called “Grid” which is read/write, and one method “Solve”.

The Grid property is defined as a long, but the values assigned to it will actually be pointers of type (CGridDef*). When assigned a new value, the pointer will be stored for later use. When retrieving the value of this property, the stored pointer must be cast to a long. I.e. Result = (long)GridPointer; or something similar.

The Solve method will be used to create a routing. The routing algorithm will be an A* line-probe algorithm using the sum of the coordinate differences as the distance. I.e. Dist=abs(x1-x2)+abs(y1-y2); computes the distance between the two points (x1,y1) and (x2,y2). (AKA delta-X plus delta-Y).

For the name of the ActiveX, use your own name, followed by the digit 2.

(Maurer2 in my case.)

Once you create your control, you use it in a visual basic program using the following code (which should be activated by a button click.)

    Maurer21.Solve

The GridDef component can be obtained from my website, as can the .h and .cpp files defining the type CGridDef. These files are called GridObj.h and GridObj.cpp, and can be found at the following URL. This URL also gives you two completed projects showing how the GridObj classes are used.

http://www.csee.usf.edu/~maurer/courses.html#UGDA

The GridDef component will be automatically installed on your computer when you access the following URL.

http://www.csee.usf.edu/~maurer/VDAL/gridroute/

Documentation for the classes in GridObj.h follows.
enum RouteType // Used in CRGrid to indicate what is where
{
    RouteNone = 0,     // Nothing Here
    RouteStart = 1,    // Starting Point
    RouteEnd = 2,      // Ending Point
    RouteObstacle = 4, // Obstacle
    RouteProbe = 8,    // Probe from starting point
    RouteRoute = 16,   // Routing
    RouteProbe2 = 32   // Probe from ending point
};

class CEdge // You don’t need to know about this one.
{
public:
    CEdge * Next;
    long StartX;
    long StartY;
    long EndX;
    long EndY;
    CEdge();
    CEdge(const CEdge &x);
    CEdge(long SX,long SY,long EX,long EY);
    CEdge &operator=(const CEdge &x);
    virtual ~CEdge();
};

class CGridDef // You interact with this to display the routing
{
public:
    void ClearEnd(void);   // Erase the end point
    void ClearStart(void); // Erase the start point
    void ClearObstacle(void); // Clear all obstacles
    void ClearProbe(void);  // Clear all probes
    void ClearRoute(void);  // Clear the routing
    void Clear(void);       // Clear everything
    long GetMaxY(void);     // Maximum X of any object in routing
    long GetMaxX(void);     // Maximum Y of any object in routing
    void DeleteProbe(CEdge * Item); // Delete a specific probe
    void NewProbe(long SX,long SY,long EX,long EY); // Add a new probe
    void DeleteRouting(CEdge * Item); // delete a routing segment
    void NewRouting(long SX,long SY,long EX,long EY); // add a routing segment
    void DeleteObstacle(CEdge * Item); // delete a specific obstacle
    void NewObstacle(long SX,long SY,long EX,long EY); // add a new obstacle
    CEdge * ProbeHead; // you don’t need to know about the rest of this stuff.
    CEdge * RoutingHead;
CEdge * ObstacleHead;
long EndY;
long EndX;
long StartY;
long StartX;
long Delta;
long Height;
long Width;
CGridDef();
CGridDef(const CGridDef &x);
CGridDef &operator=(const CGridDef &x);
virtual ~CGridDef();

};

class CRGrid // this is a more convenient representation of the routing problem
    // for internal purposes. (CgridDef is for displays.)
{
public:
    void AddGrid(CGridDef * Grid); // Copy all from CgridDef to CRGrid
    long GetCell(long X,long Y);  // Get the contents of a specific cell
    long GetStart(long &X,long &Y); // Find the start point
    long GetEnd(long &X,long &Y); // Find the end point
    long AddObstacle(long SX,long SY,long EX,long EY); // Add a new obstacle
    long AddProbe(long SX,long SY,long EX,long EY); // Add a probe (start)
    long AddProbe(long SX,long SY,long EX,long EY,long Val); // Add a probe, val determines start/end
    long AddProbe2(long SX,long SY,long EX,long EY); // add probe from end
    long AddRoute(long SX,long SY,long EX,long EY); // add routing edge
    long AddEnd(long X,long Y); // add end point
    long AddStart(long X,long Y); // add start point
    void SetDimensions(long X, long Y); // set grid dimensions (destroys everything)
    CRGrid(); // you don’t need to know about the rest
    CRGrid(CGridDef *Grid);
    virtual ~CRGrid();

private:
    long Width;
    long Height;
    long ** Data;
};

class CGridPoint // this is what is placed on the queue
{
public:
    CGridPoint * Parent;  // The parent of the current point. (You add it)
CGridPoint * Next;
long Y;
long X;
long Dir;     // use for whatever you want
CGridPoint();
CGridPoint(long NX,long NY);     // new with Dir = 0
CGridPoint(long NX,long NY,long NDir); // new with specified Dir
virtual ~CGridPoint();

class CPointQueue  // Push points on queue, pop to process. Pop moves point to the
"DONE" queue. Use the Done Queue for routing. (Chase Parent of current point)
{
public:
   virtual CGridPoint * Push(long NX,long NY); // create and push Dir=0
   virtual CGridPoint * Push(long NX,long NY,long ND); // same Dir specified.
   void Pop(CGridPoint *&GP); // returns the whole point by copying into pointer.

   CGridPoint * GetDoneHead(void); // get head of Done queue
   CGridPoint * AddDone(long X,long Y); // explicit add to done queue, Dir=0
   CGridPoint * AddDone(long X,long Y,long D); // explicit add with Dir spec.
   CGridPoint * GetContainer(long X, long Y); // Find probe with specified point.
   CGridPoint * GetContainer(long X, long Y, long D); // Same but Dir must be correct

   void AddEnd(long X, long Y); // Add end point for A* algorithms

   void Clear(void); // Erase all internal contents

   CPointQueue();     // You don’t need to know about the rest.
   virtual ~CPointQueue();
}

protected:
   BOOL OnSeg(long PX,long PY,long SX,long SY,long EX,long EY);
   CGridPoint * Done;
   CGridPoint * Tail;
   CGridPoint * Head;
   long EndX;
   long EndY;

};

class CPointQueueEuclidean : public CPointQueue
// Classes derived from CPointQueue are used for A* algorithms. The Push routines are
// changed to push points onto the queue in preferred order. Functions are overrides of
// CPointQueue functions.
public:
    virtual CGridPoint * Push(long NX, long NY);
    virtual CGridPoint * Push(long NX, long NY, long ND);
    CPointQueueEuclidean();
    virtual ~CPointQueueEuclidean();
private:
    void InsertPoint(CGridPoint * NewTemp);
};

class CPointQueueMax : public CPointQueue  // Max of delta-X and delta-Y
{
public:
    virtual CGridPoint * Push(long NX, long NY);
    virtual CGridPoint * Push(long NX, long NY, long ND);
    CPointQueueMax();
    virtual ~CPointQueueMax();
private:
    void InsertPoint(CGridPoint * NewTemp);
};

class CPointQueueMin : public CPointQueue  // Min of delta-X and delta-Y (weird!)
{
public:
    virtual CGridPoint * Push(long NX, long NY);
    virtual CGridPoint * Push(long NX, long NY, long ND);
    CPointQueueMin();
    virtual ~CPointQueueMin();
private:
    void InsertPoint(CGridPoint * NewTemp);
};

class CPointQueueSum : public CPointQueue  // sum of delta-X and delta-Y
{
public:
    virtual CGridPoint * Push(long NX, long NY);
    virtual CGridPoint * Push(long NX, long NY, long ND);
    CPointQueueSum();
    virtual ~CPointQueueSum();
private:
    void InsertPoint(CGridPoint * NewTemp);
};