Parking lot simulation.

This program creates two alleys (stacks) to park and retrieve cars.

CarNode : A class for one node of linked list. One node represents a car and its relative position in the parking alley.

Alley: A class for the stack of nodes. Two instantiations A and B are used.

```
#define MAXSIZE 5

class CarNode {

public:
     CarNode() : m_pNext(0), m_ticketNum(0) { }
     ~CarNode();
     CarNode(CarNode &) m_pNext(0), m_ticketNum(0) { }

     // assign next pointer
     void SetNext(CarNode* p){m_pNext=p;}

     // assign ticket number
     void SetTicketNum(int tN){m_ticketNum=tN;}

     // get the next pointer
     CarNode *GetNext(void){return(m_pNext);}

     // get the ticket number
     int GetTicketNum(void){return(m_ticketNum);}

private:



     int m_ticketNum;    // ticket number of car
     CarNode *m_pNext;   // pointer to next node in stack
};
```



```
class CAlley {
public:
     CAlley () : m_pTop(0), mSize(0), mMaxSize(MAXSIZE) { }
     ~CAlley () {}
     CAlley (CAlley &):m_pTop(0), mSize(0), mMaxSize(MAXSIZE) { }


     int  Park(int);               // park a car
     void Retrieve(int,CStack *);  // retrieve a car
     void Terminate();             // quit the program
     void Display(char *);         // display contents af alley


private:

     void SetTop(CarNode *p){m_pTop=p;} // assign top pointer

     // check if stack is empty
     bool Empty(){return ((mSize==0) ? true : false);}

     // check if stack is full
     bool Full() {return ((mSize==MAXSIZE) ? true : false);}
     int Push(CarNode *);  // push one node onto top of stack
     CarNode * Pop();      // pop one node from the top of stack


     CarNode *m_pTop;      // pointer to top of Allay (stack)
     int mSize;            // number of nodes in Allay (stack)
     int mMaxSize;         //max number of nodes in Allay (stack)
};


////////////////////////////////////////////////////////////
// Function: CAlley::Push
// Purpose: Add a new node to top of stack
// Parameters:
// CarNode * pNewNode- the node to be added to top of stack
// Local Variables:
//   status           - return 1 if pushed sucessfully
//                    - return 0 if stack was full
////////////////////////////////////////////////////////////

int CAlley::Push(CarNode* pNewNode)
{

}
```



```
//////////////////////////////////////////////////////////////
// Function: CAlley::Pop
// Purpose: Remove a node to top of Allay (stack).
// Parameters:
// CarNode * pNewNode- returns the node removed from top of Allay
//                     is zero if stack is empty
// Local Variables:

//////////////////////////////////////////////////////////////
CarNode * CAlley::Pop()
{

}

//////////////////////////////////////////////////////////////
// Function: CAlley::Park ( )
// Purpose: Park a car, if lot is not full. First allocate a
//          node, then add it to the top of the stack
// Parameters:
//   userTicketNum    - the ticket number for the node to be added
// Local Variables:
//   CarNode *pNewNode   - local pointer to newly allocated node
//   int status          - 1 if parked sucessfully (lot not full)
//                         0 if not parked (lot was full)
//////////////////////////////////////////////////////////////
int CAlley::Park(int userTicketNum)
{

}

//////////////////////////////////////////////////////////////
// Function: CAlley:: Retrieve ( int userTicketNum, CAlley *pB)
// Purpose: Retrieve a car from alley A. Search for the car/node
// based on ticket num. by driving a car (popping off top) out of
// A and driving (pushing onto top) into B.
// If the car is found, it is not pushed onto B, but rather,
// it is deleted. Then the cars in B are moved back into A.
//
// Parameters:
//   userTicketNum    - the ticket number for the node to be added
//   pB  -              pointer to CAlley B
//
// Local Variables:
//   CarNode *pCurr   - local pointer used as index
//  int found         - 1 if car is found, 0 if not found
//////////////////////////////////////////////////////////////
void CAlley::Retrieve(int userTicketNum, CAlley *pB)
{

}
```