CPS 100 EXAM 1 HANDOUT NAME:

Problem 8 Array A has 10 rows and 9 columns.

	0	1	2	3	4	5	6	7	8
0	.	.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.
4	.	.	.	*	*	*	*	*	.
5	.	.	*	*	o	*	G	*	.
6	.	.	*	.	.	.	.	*	.
7	.	*	.	.	.	*	.	*	.
8	.	*	*	*	*	*	*	*	.
9	.	.	.	.	.	.	.	.	.

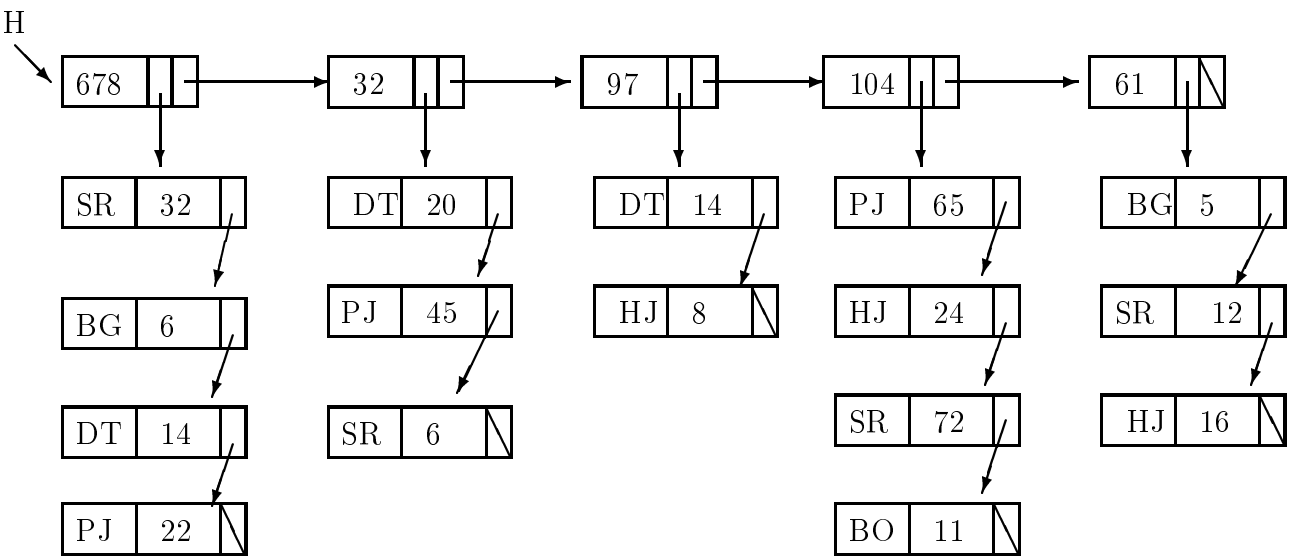
The function Move

```
const int TRUE = 1;
const int FALSE = 0;

char A[10][9];

int Move(int row, int col)
// recursive function of moves
{
   cout << "row:  " << row << "  col: " << col << endl;
   if (A[row][col]=='G') {
     cout << "   Gold found! " << endl;
     return TRUE;
    }
   else if (A[row][col] != '*') {
      A[row][col] = '*';
      if (Move(row-1,col))
         return TRUE;
      if (Move(row,col+1))
         return TRUE;
      if (Move(row+1,col))
         return TRUE;
   }
   return FALSE;
}
```

Problem 9

Consider the data structure shown below to represent the number of votes for candidates in different voting regions. There is a pointer H to the head of the list of region nodes. Each region node (rnode) contains the number of the region, a pointer to a list of candidates, and a pointer to another region. Each candidate node (cnode) contains the name of the candidate, the number of votes the candidate received in that region, and a pointer to another candidate node.

```
struct rnode {
   int Region;
   struct  cnode  * List;
   struct  rnode  * Next;
   }

struct cnode {
   String Name;
   int Votes;
   struct  cnode * Next;
   }
```



Each candidate appears at most once in each region (and may not appear), and each region appears once in the list of region nodes. None of the candidate lists or the region list are in any order.

(next page for more on Problem 9)

Problem 9c

For example, CombineRegions(97,61,H) would combine the regions 97 and 61 into the region 97. The new 97 region would have a list of 4 cnodes (the two HJ nodes would be combined into 1 node with 16+8=24 votes). CombineRegions(61,97,H) would combine regions 61 and 97 into one new region called 61.

You may assume the function DeleteRegion exists (you do not have to show the code for this function). DeleteRegion has two arguments (a region number and a pointer to the data structure). This function deletes the region node with number regnumber and deletes all the candidate nodes in its region. The function prototype is:

```
void DeleteRegion(int regnumber, struct rnode * H);
```

For example, DeleteRegion(97,H) deletes the region node with 97 from the linked list of region nodes, and deletes the two candidate nodes (DT,14) and (HJ,8).

Problem 9d, Partial Sample Output

```
Region: 61
  BG: 5 votes
  SR: 12 votes
  HJ: 16 votes
Region: 104
  PJ: 65 votes
  HJ: 24 votes
...
```

Problem 10

Here is the Stack Class we saw in the book.

```
struct stackNode;  // defined in implementation file
typedef stackNode* ptrType;  // pointer to node

enum boolean {FALSE,TRUE};

class stackClass
{
public:
// constructors and destructor:
   stackClass();                      // default constructor
   stackClass(const stackClass& S);  // copy constructor
```

```
   ~stackClass();                    // destructor

// stack operations:
   boolean StackIsEmpty();
   // Determines whether a stack is empty.
   // Precondition: The constructor has been called.
   // Postcondition: Returns TRUE if the stack was
   // empty, otherwise returns FALSE.

   void Push(stackItemType NewItem, boolean& Success);
   // Adds an item to the top of a stack.
   // Precondition: The constructor has been called. NewItem
   // is the item to be added.
   // Postcondition: If insertion was successful, NewItem
   // is on the top of the stack and Success is TRUE;
   // otherwise Success is FALSE.

   void Pop(boolean& Success);
   // Removes the top of a stack.
   // Precondition: The constructor has been called.
   // Postcondition: If the stack was not empty, the item
   // that was added most recently is removed and Success
   // is TRUE. However, if the stack was empty, deletion is
   // impossible and Success is FALSE.

   void GetStackTop(stackItemType& StackTop,
    boolean& Success);
   // Retrieves the top of a stack.
   // Precondition: The constructor has been called.
   // Postcondition: If the stack was not empty, StackTop
   // contains the item that was added most recently and
   // Success is TRUE. However, if the stack was empty, the
   // operation fails, StackTop is unchanged, and Success
   // is FALSE. The stack is unchanged.

private:
   ptrType TopPtr;  // points to top of stack
};
```

(extra page)