```c
/*
 * mallocMatrix  returns dbl pointer to a mallocted matrix, need dimensons rows and columns
 * and an name just vor better debugging -- values are stored in the matrix structure
 *
 */


int **mallocMatrix(matrix matrixTemp) {

  int **matrixT;
  matrixT=(int **) malloc(matrixTemp.rows*sizeof(int *)); // malloc r times the space for an int array
                                                 // (in every row will be one = thats are the coloums
  if(NULL == matrixT){
    printf(" No memory available for matrix %s!\n\n",matrixTemp.name);
    exit( EXIT_FAILURE);
  }
  for(row=0;row<matrixTemp.rows;row++){ // go throug every row
    matrixT[row] = (int *) malloc(matrixTemp.cols * sizeof(int));  // allocate buffer for c times the
space of
                                                    // an int --> we store int values in our matrices
    if(NULL == matrixT[row]){
      printf(" No memory for matrix %s row %d\n\n",matrixTemp.name,row);
     exit( EXIT_FAILURE);
    }
  }
  printf(" Malloc for matrix %s finished\n",matrixTemp.name);

  return (matrixT); // returns the pointer to matrix

}

/*
 * freeMatrix deallocate buffer from passed matrix
 */

void freeMatrix(matrix matrixDel){

        for(row = 0; row < matrixDel.rows; row++){ // oposite way the malloc first go throug every row
            free(matrixDel.matrix[row]);     // and free every column
        }
        free(matrixDel.matrix); // then free all rows of matrix
}

/*
 * printMatrix  needs a dpl pointer to matrix  r an c are dimensions of the passed matrix,
 * option for the way to print, matrixname for print out all matrix data now in one datatype
 *
 */

void printMatrix(matrix tempMatrix, int option){

    if(option>=2){ // print to file, print to file and screen

          char filename[40];
          //print int to string, so it can be used as unique filename
          snprintf(filename, sizeof(filename), "log/Result_LogID_%d.txt", (int)timestamp);

          FILE *file; // point to file
          if((file= fopen(filename,"a"))==NULL) // open file and add at content at the end of file,
                                                // --> so all 3 matricies are in one file
          {
            printf("\nCan't open the file!\n\n ");
          }else{
            //print name and dimensions of a matrix
            fprintf(file,"\n%s[%d][%d]\n\n",tempMatrix.name,tempMatrix.rows,tempMatrix.cols);
            for(row=0;row<tempMatrix.rows;row++){
              for(col=0;col<tempMatrix.cols;col++){
                    fprintf(file,"%d;",tempMatrix.matrix[row][col]); // run throug matrix, print
values from each field
              }
              fprintf(file,"\n");
```

```
            }
            printf("\n\nValues of matrix %s stored in %s.\n",tempMatrix.name,filename);

          }
          fclose(file); //close file

          if(option==3){ // set option to 1 so it would also be printed to screen
                option=1;
          }
    }
    if(option==1){ // print so screen
            printf("\n%s[%d][%d]\n\n",tempMatrix.name,tempMatrix.rows,tempMatrix.cols);
            for(row=0;row<tempMatrix.rows;row++){
              for(col=0;col<tempMatrix.cols;col++){
                printf(" %5d",tempMatrix.matrix[row][col]);
              }
              printf("\n");
            }
    }
}

/*
 * matrixCompare compares 2 matrices wheter their values are equal or not
 *
 */


void matrixCompare(matrix matrixA, matrix matrixB){
    printf("Compare matrix %s and matrix %s.\n",matrixA.name,matrixB.name);
    if(matrixA.rows!=matrixB.rows || matrixA.cols!=matrixB.cols){
          printf("Matrix %s and matrix %s have not the same dimensions!\n
\n",matrixA.name,matrixB.name);
          return;
    }
    for(row=0;row<matrixA.rows;row++){
        for(col=0;col<matrixA.cols;col++){

        if(matrixA.matrix[row][col] != matrixB.matrix[row][col]){
          printf("Matrix %s and matrix %s have not the same value at [%d][%d]!\n
\n",matrixA.name,matrixB.name,row,col);
          return;
        }

        }
    }
  printf("Matrix %s and matrix %s are equal.\n\n",matrixA.name,matrixB.name);
  return;
}


/*
 * matrixInitRowPlusCol initialies a matrix by calculating the indexies
 *
 */

void matrixInitRowPlusCol(matrix matrixA){

  printf("\nInitialise matrix %s with %s[m][n] = m+n ...\n",matrixA.name,matrixA.name);
  for(row=0;row<matrixA.rows;row++){
    for(col=0;col<matrixA.cols;col++){
      // stored value will be calculted by row and col indexies,
      matrixA.matrix[row][col] = row + col; // but any thing else would be
possilble
    }
  }
  printf("finished\n");
}


/*
 * matrixInitRandom initialies a matrix with random values from 0 to 100
 *
```

```
 */

void matrixInitRandom(matrix matrixA){

  printf("\nInitialise matrix %s with %s[m][n] = random ...\n",matrixA.name,matrixA.name);
  for(row=0;row<matrixA.rows;row++){
    for(col=0;col<matrixA.cols;col++){
      matrixA.matrix[row][col] = rand()%100; // stored value will be random between 0-100
    }
  }
  printf("finished\n");
}
```