```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    //variable declarations
    int **ptr_iArray = NULL; //A pointer-to-pointer to integer ... but can also hold base address of a 2D Array which will can have any number of rows and any number of columns ...
    int i, j;
    int num_rows, num_columns;

    //code

    // *** ACCEPT NUMBER OF ROWS 'num_rows' FROM USER ***
    printf("\n\n");
    printf("Enter Number Of Rows : ");
    scanf("%d", &num_rows);

    // *** ACCEPT NUMBER OF COLUMNS 'num_columns' FROM USER ***
    printf("\n\n");
    printf("Enter Number Of Columns : ");
    scanf("%d", &num_columns);

    // *** ALLOCATING MEMORY TO 1D ARRAY CONSISTING OF BASE ADDRESS OF ROWS ***
    printf("\n\n");
    printf("********** MEMORY ALLOCATION TO 2D INTEGER ARRAY **********\n\n");
    ptr_iArray = (int **)malloc(num_rows * sizeof(int *));
    if (ptr_iArray == NULL)
    {
        printf("FAILED TO ALLOCATE MEMORY TO %d ROWS OF 2D INTEGER ARRAY !!! EXITTING NOW...\n\n", num_rows);
        exit(0);
    }
    else
        printf("MEMORY ALLOCATION TO %d ROWS OF 2D INTEGER ARRAY SUCCEEDED !!! \n\n", num_rows);

    // *** ALLOCATING MEMORY TO EACH ROW WHICH IS A 1D ARRAY CONTAINING CONSISTING OF COLUMNS WHICH CONTAIN THE ACTUAL INTEGERS ***
    for (i = 0; i < num_rows; i++)
    {
        ptr_iArray[i] = (int *)malloc(num_columns * sizeof(int)); //ALLOCATING MEMORY (Number Of Columns * size of 'int') TO ROW 'i'
        if (ptr_iArray[i] == NULL) //ROW 'i' MEMORY ALLOCATED ?
        {
            printf("FAILED TO ALLOCATE MEMORY TO COLUMNS OF ROW %d OF 2D INTEGER ARRAY !!! EXITTING NOW...\n\n", i);
            exit(0);
        }
        else
            printf("MEMORY ALLOCATION TO COLUMNS OF ROW %d OF 2D INTEGER ARRAY SUCCEEDED !!!\n\n", i);
    }

    // *** FILLING UP VALUES ***
```

```
49      for (i = 0; i < num_rows; i++)
50      {
51          for (j = 0; j < num_columns; j++)
52          {
53              ptr_iArray[i][j] = (i * 1) + (j * 1); // can also use : *(*
                  (ptr_iArray + i) + j) = (i * 1) + (j * 1)
54          }
55      }
56
57      // *** DISPLAYING VALUES ***
58      for (i = 0; i < num_rows; i++)
59      {
60          printf("Base Address Of Row %d : ptr_iArray[%d] = %p \t At Address : %p
                \n", i, i, ptr_iArray[i], &ptr_iArray[i]);
61      }
62
63      printf("\n\n");
64
65      for (i = 0; i < num_rows; i++)
66      {
67          for (j = 0; j < num_columns; j++)
68          {
69              printf("ptr_iArray[%d][%d] = %d \t At Address : %p\n", i, j,
                    ptr_iArray[i][j], &ptr_iArray[i][j]); // can also use *(*
                    (ptr_iArray + i) + j) for value and *(ptr_iArray + i) + j for
                    address ...
70          }
71          printf("\n");
72      }
73
74      // *** FREEING MEMORY ALLOCATED TO EACH ROW ***
75      for (i = (num_rows - 1); i >= 0; i--)
76      {
77          if (ptr_iArray[i])
78          {
79              free(ptr_iArray[i]);
80              ptr_iArray[i] = NULL;
81              printf("MEMORY ALLOCATED TO ROW %d HAS BEEN SUCCESSFULLY FREED !!!
                    \n\n", i);
82          }
83      }
84
85      // *** FREEING MEMORY ALLOCATED TO 1D ARRAY CONSISTING OF BASE ADDRESSES OF
            ROWS ***
86      if (ptr_iArray)
87      {
88          free(ptr_iArray);
89          ptr_iArray = NULL;
90          printf("MEMORY ALLOCATED TO ptr_iArray HAS BEEN SUCCESSFULLY FREED !!!
                \n\n");
91      }
92
93      return(0);
94  }
95
96
```