

ECE 264 Advanced C Programming

2009/02/04

Contents

1 Memory Allocation for Matrix	1
--------------------------------	---

1 Memory Allocation for Matrix

The following example shows how to allocate memory for a 2-dimensional array (i.e. matrix). When we allocate a high-dimensional array, we have to do it **one dimension at a time**.

```
/* arraypointer.c */
/* This example is related to Sec 12.6 of ABoC. */
#include <stdlib.h>
#include <stdio.h>

void printMatrix(int * * ma, int row, int col)
{
    int rcnt;
    int ccnt;
    for (rcnt = 0; rcnt < row; rcnt++)
    {
        for (ccnt = 0; ccnt < col; ccnt++)
        {
            printf(" %5d ", ma[rcnt][ccnt]);
        }
        printf("\n");
    }
    printf("\n");
}

int main(int argc, char * argv[])
{
    int numRow;
    int numCol;
    int row;
```

```

int col;
int * * matrix;
if (argc < 3)
{
    printf("need numRow and numCol\n");
    return -1;
}
numRow = (int)strtol(argv[1], (char **)NULL, 10);
numCol = (int)strtol(argv[2], (char **)NULL, 10);
if ((numRow <= 0) || (numCol <= 0))
{
    printf("both numbers must be positive\n");
    return -1;
}
matrix = malloc(numRow * sizeof(int *));
if (matrix == 0)
{
    printf("memory allocation fail\n");
    return -1;
}
for (row = 0; row < numRow; row++)
{
    matrix[row] = malloc(numCol * sizeof(int));
    if (matrix[row] == 0)
    {
        printf("memory allocation fail\n");
        return -1;
    }
}
for (row = 0; row < numRow; row++)
{
    for (col = 0; col < numCol; col++)
    {
        matrix[row][col] = row + col;
    }
}
printMatrix(matrix, numRow, numCol);

for (row = 0; row < numRow; row++)
{
    for (col = 0; col < numCol; col++)
    {

```

```

        matrix[row][col] *= 2;
    }
}
printMatrix(matrix, numRow, numCol);

for (row = 0; row < numRow; row++)
{
    free(matrix[row]);
}
free(matrix);
return 0;
}

```

Each matrix is a pointer of arrays. Since each array is also a pointer, a matrix is declared as pointers to pointers

```
int * * matrix;
```

This program first creates an array of **pointers**:

```
matrix = malloc(numRow * sizeof (int *));
```

Notice `int *` inside `sizeof`. Then it allocates an array for each pointer. Each row points to a piece of memory for storing the elements:

```

for (row = 0; row < numRow; row++)
{
    matrix[row] = malloc(numCol * sizeof (int));
}
```

Each element is initialized to the sum of the row and the column indexes.

```
matrix[row][col] = row + col;
```

Before the program ends, we need to release the memory. Each row is released first and then the pointers to the rows are released. **Memory allocation and release are usually symmetric.**

Exercise How to allocate space to store a list of strings? Your PA1 probably needs this.

The following means `argv` is an array of strings. Since each string is an array of characters, `argv` is a 2-dimensional array.

```
int main(int argc, char * argv[])
```