

# Microcontroller / C-Programmierung

## Selbststudium Semesterwoche 6

### 1. Variable Parameterliste und Matrixen (pointer-to-pointer)

Schreiben Sie die Funktion `createMatrix(...)`, welche eine Matrix erzeugt und die Matrix mit einer beliebigen Anzahl Werte initialisiert (die Anzahl wird als dritter Übergabeparameter übergeben). Alle anderen Werte in der Matrix sollen auf 0 gesetzt werden. Der untenstehende Aufruf erzeugt eine 3x4 Matrix mit dem Inhalt

|     |     |     |     |
|-----|-----|-----|-----|
| 111 | 222 | 333 | 444 |
| 555 | 666 | 0   | 0   |
| 0   | 0   | 0   | 0   |

```
int **m;
m = createMatrix(3, 4, 6, 111, 222, 333, 444, 555, 666);
```

1. Parameter: Anzahl Zeilen
2. Parameter: Anzahl Spalten
3. Parameter: Anzahl Werte (Anzahl Argumente in der variablen Parameterliste)
4. Parameter: Initialisierungswerte für die Matrix (Anzahl entspricht dem 3. Parameter)

**Hinweis:** Gehen Sie iterativ vor! Konzentrieren Sie sich zuerst auf das dynamische allozieren einer Matrix. Setzen Sie vorerst alle Werte auf 0. Erst wenn diese Teilfunktion läuft und getestet ist, fügen Sie die variable Argumentliste hinzu.

```
// Iteration 1: Variable Argument List
#include <stdio.h>
#include <stdarg.h> // contains a set of macro definitions
// that define how to step through an argument list.

void minprintf(char *fmt, ...);

void main (void)
{
    char *arg1 = "%d wird ersetzt mit arg2 (10)..";
    int *arg2 = 10;
    minprintf(arg1, arg2);
    getchar();
}

// minprintf: minimal printf with variable argument list
void minprintf(char *fmt, ...)
{
    va_list ap; // points to each unnamed arg in turn
    char *p, *sval;
    int ival;
    double dval;
    va_start(ap, fmt); // make ap point to 1st unnamed arg
    for (p = fmt; *p; p++) {
        if (*p != '%') {
            putchar(*p);
            continue;
        }
        switch (*++p) {
            case 'd':
                ival = va_arg(ap, int);
                printf("%d", ival);
                break;
            case 'f':
                dval = va_arg(ap, double);
                printf("%f", dval);
```

```

        break;
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
                putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
    }
    va_end(ap); // clean up when done
}

```

### // Iteration 2: VA list in conjunction with dynamically allocated matrices

```

#include <stdio.h>
#include <stdarg.h> // contains a set of macro definitions
                  // that define how to step through an argument list.

int** createMatrix(int rows, int cols, int count, ...);
void printMatrix(int **rowptr, int rows, int cols);
void displayMemStat(int **rowptr, int rows);

void main (void)
{
    int **m; // points to a int pointer
            // **m is actually the top-left integer element of the matrix
    int rows = 3;
    int cols = 4;
    int values = 6;

    // create a new 3x4 matrix with 6 values (111,222,333,444,555 and 666)
    m = createMatrix(rows, cols, values, 111, 222, 333, 444, 555, 666);

    // print the matrix on the console
    printMatrix(m, rows, cols, cols);
    printf("\n\n");

    // display a simple memory static
    displayMemStat(m, rows);

    getchar();
}

int** createMatrix(int rows, int cols, int count, ...)
{
    int row, col;
    int **rowptr;

    char *p, *sval;
    int ival = 0;
    double dval;

    // argument pointer ap points to each unnamed arg in turn
    va_list ap;

    // count is the last named argument in the list
    // make ap point to 1st unnamed arg
    va_start(ap, count);

    // allocate memory for rows
    rowptr = malloc(rows * sizeof(int *));
    if (rowptr == NULL)
    {
        puts("\nFailed to allocate memory for row pointers.\n");
        return 0;
    }
}

```

```
// allocate memory for columns
for (row = 0; row < rows; row++)
{
    rowptr[row] = malloc(cols * sizeof(int));
    if (rowptr[row] == NULL)
    {
        printf("\nFailed to allocate memory for row[%d]\n",row);
        return 0;
    }
}

// insert values into the appropriate cells
for (row = 0; row < rows; row++)
{
    for (col = 0; col < cols; col++)
    {
        if(count-->0)
        {
            rowptr[row][col] = va_arg(ap, int);
        }
        else
        {
            rowptr[row][col] = 0;
        }
    }
}

va_end(ap); // clean up argument list
return rowptr;
}

void printMatrix(int **rowptr, int rows, int cols)
{
    int row = 0, col = 0;

    printf("%dx%d MATRIX\n", rows, cols);
    printf("=====\n");
    for (row = 0; row < rows; row++)
    {
        for (col = 0; col < cols; col++)
        {
            printf("%i\t ", rowptr[row][col]);
        }
        printf("\n");
    }
}

void displayMemStat(int **rowptr, int rows)
{
    int row = 0;
    printf("MEMORY STATISTIC\n");
    printf("=====\n");
    printf("Row\tPointer(hex)\tPointer(dec)\tDiff(dec)");
    for (row = 0; row < rows; row++)
    {
        printf("\n%d\t%p\t%d", row, rowptr[row], rowptr[row]);
        if (row > 0)
        {
            printf("\t\t%d", (int)(rowptr[row] - rowptr[row-1]));
        }
    }
}
```