Inhalt

11 Dynamische Feldvereinbarung

In der Regel ist die Anzahl der notwendigen Feldkomponenten eines Feldes zum Zeitpunkt der Programmierung nicht bekannt. Dann muss die Speicherzuordnung während der Laufzeit des Programms erfolgen.
In der Standardbibliothek **<stdlib.h>** werden Funktionen zur dynamischen **Speicherplatzvereinbarung** während der Laufzeit im sogenannten **Heap** (Haufen) deklariert. Eine Funktion gestattet die **Bereitstellung** von Speicherplatz.

Speicherplatzbereitstellung: **void * malloc (** *size_t* **) ;**

Die Funktion **malloc** liefert einen Zeiger auf einen Speicherbereich von der angegebenen Größe. Falls kein Speicher zur Verfügung gestellt werden kann, liefert sie den **NULL**-Zeiger zurück.

void *:
- Zeiger ohne Typ, deshalb nicht dereferenzierbar, kompatibel zu allen anderen Zeigertypen.
- Typumwandlung notwendig.

*size_t***:**
- *size_t* wird in <stdio.h> definiert:

```
#ifndef _SIZE_T
#define _SIZE_T
typedef unsigned long size_t;
#endif
```

- Vorzeichenloser ganzzahliger *Typ*, dessen Wertebereich alle zulässigen Speicherbereichsgrößen in Byte umfasst.
- Der unäre Operator **sizeof (** *Typ* **) ;** ermittelt die Anzahl der benötigten Bytes einer Größe mit dem angegebenen *Typ*. Er liefert folglich einen Wert vom Typ *size_t*.

```
double *z; z = ( double *) malloc( sizeof( double));
```

Hier wird im Heap ein Speicherbereich für einen *double_Wert* bereitgestellt und ein Zeiger darauf zurückgeliefert.

Eine weitere Funktion gestattet die **Freigabe** des bereitgestellten Speicherplatzes.

Speicherplatzfreigabe: **void free (void *) ;**

- Nicht mehr benötigter Speicher eines durch **malloc** erzeugten Zeigers wird wieder freigegeben.
- Nicht mehr benötigter Speicherplatz wird auch, ohne **free**, nach Programmablauf freigegeben (Speicherplatzbedarf!).

```
free( z );
```

11.1 Dynamische Vereinbarung von Vektoren

V: Zeiger auf ein Feld von Feldkomponenten

Feldkomponente: $v_i \Rightarrow V[i]$

Standardoperationen

Feld erzeugen: **`double *initVektor( unsigned int);`**
Feld freigeben: **`void freeVektor(double *);`**

vektor.h

```
/*
 * Header zum Modul vektor.c:
 * Zugriff auf Vektoren ueber Zeiger
 */

/*
 * Erzeugt dynamisch einen Vektor
 * Parameter: Komponentenzahl
 * Rueckgabe: Zeiger auf Vektor
 */
double *initVektor( unsigned int);

/*
 * Loescht ein dynamisch erzeugten Vektor
 * Parameter: Zeiger auf Vektor
 */
void freeVektor( double *);
```

vektor.c

```
/*
 * Implementation zum Header vektor.h:
 * Zugriff auf Vektoren ueber Zeiger
 */
# include <stdlib.h>
# include "vektor.h"

double *initVektor( unsigned int Komponentenzahl)
{
  return
  (double *) malloc( sizeof(double) * Komponentenzahl);
}

void freeVektor( double *Vektor)
{
  free( Vektor);
  return;
}
```

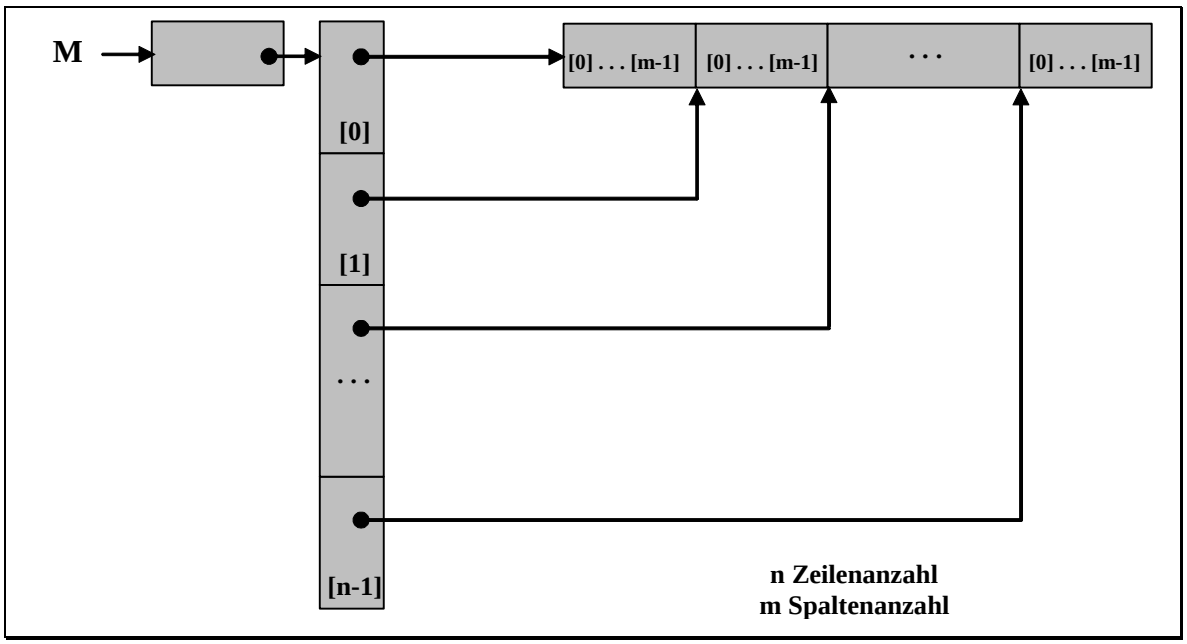
myvektor.c

```
/*
 * Testprogramm dynamisch vereinbarte Vektoren
 * myvektor.c:
 * Anlegen und Freigeben eines Vektors,
 * Schreiben und Lesen von Vektorkomponenten
 * unter Verwendung des Moduls vektor.h/vektor.c
 */

# include <stdio.h>
# include "vektor.h"

int main()
{
  double *V; unsigned int i, n;

  printf( "Eingabe Komponentenzahl n\n");
  printf( "n="); scanf( "%i", &n);

  V = initVektor( n);       /* Anlegen eines Vektors */

  printf( "Schreiben:\n");  /* Schreiben des Vektors */
  for( i = 0; i < n; i++) V[ i] = i;

  printf( "Lesen:\n");          /* Lesen des Vektors */
  for( i = 0; i < n; i++)
    printf( "V[ %d] = %f\n", i, V[ i]);

  freeVektor( V);           /* Freigeben des Vektors */
  return 0;
}
```

makefile

```
# makefile fuer myvektor

CC = gcc
CFLAGS = -ansi -Wall -O -c
OBJ = myvektor.o vektor.o

myvektor: $(OBJ) vektor.h makefile
   $(CC) $(OBJ) -o myvektor

myvektor.o: myvektor.c vektor.h
   $(CC) $(CFLAGS) myvektor.c

vektor.o: vektor.c vektor.h
   $(CC) $(CFLAGS) vektor.c

clean:
   rm -f $(OBJ)
```

11.2 Dynamische Vereinbarung von Matrizen



M: Zeiger auf ein Feld von Zeigern auf die Matrixkomponenten

Matrixkomponente: $m_{ij} \Rightarrow M[i][j]$

Standardoperationen

Matrix erzeugen:

```
double **initMatrix( unsigned int, unsigned int);
```

Matrix freigeben:

```
void freeMatrix(double **);
```

matrix.h

```
/*
 * Header zum Modul matrix.c:
 * Zugriff auf Matrizen ueber Zeiger auf Zeiger
 */

/*
 * Erzeugt dynamisch eine Matrix
 * Parameter: Zeilenzahl, Spaltenzahl
 * Rueckgabe: Zeiger auf Matrix/
 */
double **initMatrix( unsigned int, unsigned int);

/*
 * Loescht eine dynamisch erzeugte Matrix
 * Parameter: Zeiger auf Matrix
 */
void freeMatrix( double **);
```

matrix.c

```
/*
 * Implementation zum Header matrix.c:
 * Zugriff auf Matrizen ueber Zeiger auf Zeiger
 */

# include <stdlib.h>
# include "matrix.h"

double **initMatrix
  ( unsigned int Zeilenzahl, unsigned int Spaltenzahl)
{
  double **Matrix, *p; int i;

            /* Zeiger auf Feld von Zeigern ( Zeilen )*/
  Matrix = ( double **) malloc
                      ( sizeof( double*) * Zeilenzahl);

                                /* Zeiger auf Matrix, */
           /*zeilenweise hintereinander abgespeichert*/
  Matrix[ 0] = ( double *) malloc
         ( sizeof( double) * Zeilenzahl * Spaltenzahl);

                                /* Zeiger auf 2.Zeile */
  p = Matrix[0] + Spaltenzahl;
                /* Eintrag der Zeiger auf die Zeilen */
  for( i = 1; i < Zeilenzahl; i++, p += Spaltenzahl)
    Matrix[ i] = p;

  return Matrix;
}


void freeMatrix( double **Matrix)
{
  free( *Matrix);
  free( Matrix);
  return;
}
```

mymatrix.c

```
/*
 * Testprogramm
 *      Zugriff auf Matrizen ueber Zeiger auf Zeiger
 * mymatrix.c: Anlegen und Freigeben einer Matrix,
 *     Schreiben und Lesen von Matrixelementen
 *     unter Verwendung des Moduls matrix.h/matrix.c
 */

# include <stdio.h>
# include "matrix.h"

int main()
```

```
{
  double **M;

  M = initMatrix( 3, 3);      /* Anlegen einer Matrix */

                             /* Schreiben in die Matrix */
  printf( "Schreiben: M[ 1, 2] = 3\n"); M[ 1][ 2] = 3;

                               /* Lesen aus der Matrix */
  printf( "Lesen:     M[ 1, 2] = %f\n", M[ 1][ 2]);

  freeMatrix( M);           /* Freigeben einer Matrix */

  return 0;
}
```

makefile

```
# makefile fuer mymatrix

CC = gcc
CFLAGS = -ansi -Wall -O -c
OBJ = mymatrix.o matrix.o

mymatrix: $(OBJ) matrix.h makefile
   $(CC) $(OBJ) -o mymatrix

mymatrix.o: mymatrix.c matrix.h
   $(CC) $(CFLAGS) mymatrix.c

matrix.o: matrix.c matrix.h
   $(CC) $(CFLAGS) matrix.c

clean:
   rm –f $(OBJ)
```

11.3 Die Kommandozeile

Der Standard sieht vor, dass sich ein Programm über das Betriebssystem eine Reihe von Zeichenketten besorgen kann, die beim Aufruf des Programms festgelegt werden.

```
$ cp dat1 dat2
```
Kopiert Datei dat1 nach Datei dat2

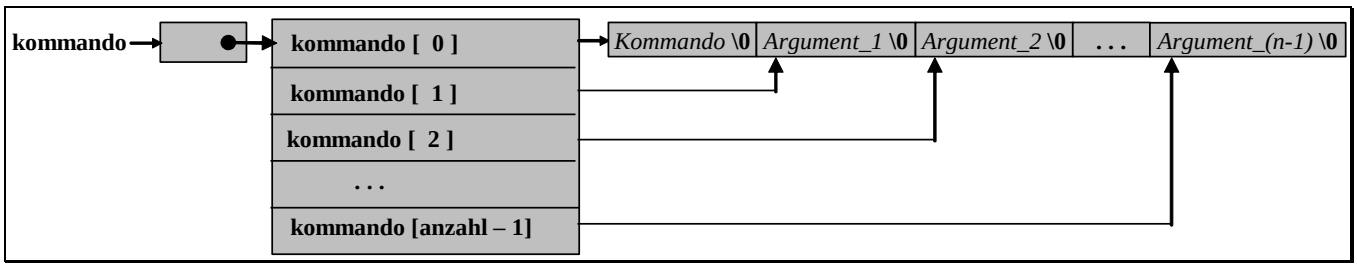
Die Kommandozeile **Kommando Argument_1 Argument_2 . . . Argument_(n-1)** wird dabei anhand der Leerzeichen in verschiedene Zeichenketten zerlegt. Zur Beschaffung dieser Zeichenketten ist es notwendig, den Hauptfunktionskopf abzuändern.

Statt: **int main ( ) { . . . }**
setzt man jetzt: **int main (int** *Variablename*, **char * *** *Zeigername***) { . . . }**

Dabei wird der *Variablen* die Anzahl der Wörter der Kommandozeile zugewiesen. Der *Zeiger* zeigt auf den Anfang des Kommandos. Sei die Variable **anzahl** und der Zeiger **kommando**:

int main (int anzahl, char * * kommando) { . . . }

Datenstruktur:



kmd.c

```
/*
 * kmd.c Auswertung einer Kommandozeile
 * Demoprogramm
 */

# include <stdio.h>

int main( int anzahl, char **kommando)
{
  int i;
  printf( "Anzahl der Argumente: %d\n", anzahl);
  for( i = 0; i < anzahl; i++)
    printf( "%d. Argument: '%s'\n", i, kommando[i]);
  return 0;
}
```

Aufruf: **kmd 1 2 3** ⇒ **Anzahl der Argumente: 4**
0. Argument: ´kmd´
1. Argument: ´1´
2. Argument: ´2´
3. Argument: ´3´