16.12 Array

Montag, 23. Mai 2022 10:32

Aufgabenstellung

Die Aufgaben sind direkt im Code enthalten. Neben dem Schreiben der Funktionen soll natürlich auch der Aufruf der Funktionen innerhalb von main() korrekt erstellt/angepasst werden. Dazu müssen z.T. die gegebenen Arrays vor den Aufruf der Testfunktionen konvertiert und anschließend wird zurückkonvertiert werden.

Hinweise:

- In der Beschreibung nutze ich das Wort Matrix als zweidimensionales Array
- Die Funktionalitäten der Funktionen sind trivial. Der Schwerpunkt liegt entsprechend der Aufgabenstellung beim Umgang mit Zeiger auf Arrays.
- Sie müssen in dieser Aufgabe des öfteren malloc() nutzen. Damit die Aufgabe nicht noch umfangreicher wird, gilt
  - keine Kontrolle notwendig, ob malloc() fehlgeschlagen ist
  - die Freigabe eines zuvor mit malloc reservierten Speichers ist nicht notwendig

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>   //fuer uintptr_t
#include <stdlib.h>   //fuer strtol()
//Compilerschalter: -Wall -fsanitize=address
typedef enum {DUMP_8, DUMP_16, DUMP_32,
              DUMP_8A,DUMP_16A,DUMP_32A,} DUMP_MODE;
int dump(void *start, size_t len, int width, DUMP_MODE mode);
int32_t mat1[4][5] = { {10, 20, 30, 40},
                       {11, 22, 33, 44},
                       {01, 02, 03, 04}};
int32_t mat2[4][5] = { {[2]=3},{[0 ... 2]=2},{[1]=7,8},[1][2]=1,
                        [3][1]=-1,[3][1 ... 2]=-2};
//Erstellen sie einen Zeiger auf ein zweidimensionales Array, welches z.B.
//den Rückgabewert von Aufgabe 1b) aufnehmen kann.
#if 0
___   ___ mat3 ___;
#endif

//Erstellen sie eine Alias eines Datentyps zur Nutzung für die Call By Value Aufgaben
#if 0
typedef __________
#endif
//Aufgabe 1a) Matrix mit beliebiger Dimension als Call by Reference übergeben
//Erstellen sie eine Funktion, welche den Inhalt einer Matrix ausgibt.
//   Die darzustellende Matrix wird als Call By Reference übergeben. Die Dimension
//   der Matrix wird ebenfalls übergeben, so dass diese Funktion beliebige Matrizen
//   darstellen kann.
//Aufruf/Test der Funktionalität bspw. mittels:
//>>debugr [mat1|mat2|mat3] dim1 dim2
//>>debugr mat1 4 5
//>>debugr mat1 4 4
//>>debugr mat1 5 4
//>>debugr mat2 1 20
//Tipp: Bevor sie sich den Inhalt von mat1/mat2 anschauen, versuchen sie den Inhalt
//      der Matrix auf Basis der Initialisierung zu ermitteln
#if 0
void debugr( ___   arr  ___ )
{
    if(arr==NULL)
        printf("Array nicht definiert!\n");
    else {
    }
}
#endif
```

```
//Aufgabe 1b) Matrizen mit fester Dimension als Call by Reference über- und zurückgeben
//Erstellen sie ein Funktion, welche zwei Matrizen subtrahiert und die
//   resultierende Matrix als Rückgabeparameter an den Aufrufer zurückgibt.
//   Benutzend sie bei der Datentypdefinition für den minuend und den subtrahend
//   jeweils ein andere Schreibweise!
//Aufruf/Test der Funktionalität bspw. mittels
//>>sub [mat1|mat2|mat3] [mat1|mat2|mat3]
//>>sub mat1 mat2
//>>sub mat1 mat1
#if 0
 ___ sub( ___ minued ___  , ___ subtrahend ___)
{
   ___  diff ___ =malloc( ___ );
    return diff;
}
#endif
//Aufgabe 2a) Matrix per Call By Value übergeben
//Erstellen sie eine Funktion, welche den Inhalt einer Matrix ausgibt
//   Die darzustellende Matrix wird als Call By Value übergeben. Die
//   Dimension ist fix.
//Aufruf/Test der Funktionalität bspw. mittels
//>>debugv [mat1|mat2|mat3]
#if 0
void debugv(________ mat )
{
}
#endif
//Aufgabe 2b) Matrix per Call By Value zurückgeben
//Erstellen sie eine Funktion, welche eine Matrix fixer Größe [4][5]
//   erstellt, die einzelnen Elemente mittels matr[zeile][spale]=zeile*100+spalte;
//   initialisiert und diese als Call by Value zurückgibt
//Aufruf/Test der Funktionalität bspw. mittels
//>>allocv [mat1|mat2|mat3]
#if 0
____ allocv(void)
{
}
#endif
//Aufgabe 3a) Übergabe von Subarray
//Erstellen sie eine Funktion, welche nur eine Spalte einer Matrix
//   übergeben bekommt und diese ausgibt.
//Aufruf/Test der Funktionalität bspw. mittels
//>>spalte [mat1|mat2|mat3]  spalte
//>>spalte mat1 3
#if 0
void spalte( ___ subaray ___ )
{
}
#endif
//Aufgabe 3b) Übergabe von Subarray
//Erstellen sie eine Funktion, welche nur eine Zeile einer Matrix
//   übergeben bekommt und diese ausgibt.
//Aufruf/Test der Funktionalität bspw. mittels
//>>zeile [mat1|mat2|mat3]  spalte
//>>zeile mat1 3
#if 0
void zeile( ___ subaray ___ )
{
}
#endif
const char help[]= "\n"
  "dump   mat1|mat2                - Speicherdump einer Matrix\n"
  "debugr mat1|mat2|mat3 dim1 dim2  - Formatierte Ausgabe der Matrix\n"
  "sub    mat1|mag2 mat1|mat2       - Subtraktion mat3=1.Operand-2.Operand\n"
  "allocv mat1|mat2|mat3            - Array Initialisieren\n"
  "debugv mat1|mat2|mat3            - Formatierte Ausgabe der Matrix\n"
```

```
  "zeile  mat1|mat2|mat3 nr         - Ausgabe einer Zeile\n"
  "spalte mat1|mat2|mat3 nr         - Ausgabe einer Spalte\n"
  "help                             - Führt alle Operationen auf.\n"
  "quit                             - Beendet das Programm.\n";
int main(int argc, char const *argv[])
{
    enum status {OK,KO_FGETS,KO_END} ret=OK;
    printf("File:"__FILE__ " erstellt am "__DATE__" um "__TIME__ " gestartet\n");
    printf(help);
    while(1) {
        char input[100];
        if(fgets(input,sizeof input,stdin)==NULL) {
            ret=KO_FGETS;
            break;
        }
        char *arg1 = strtok(input, " \n\r");
        char *arg2 = strtok(NULL , " \n\r");
        char *arg3 = strtok(NULL , " \n\r");
        char *arg4 = strtok(NULL , " \n\r");
        if(arg1==NULL) {
            continue;
        }
#if 0
        else if((strcmp(arg1,"debugr")==0) && (arg2!=NULL) && (arg3!=NULL) && (arg4!=NULL)) {
            int zeilen= (int)strtol(arg3,NULL,0);
            int spalten=(int)strtol(arg4,NULL,0);
            if(strcmp(arg2,"mat1")==0)
              debugr(___  mat1  ___);
            else if(strcmp(arg2,"mat2")==0)
              debugr(___  mat1  ___);
            else if(strcmp(arg2,"mat3")==0)
              debugr(zeilen,spalten,(___)mat3);
            else
              printf("Invalid Matrix\n");
        }
        else if((strcmp(arg1,"sub")==0) && (arg2!=NULL) && (arg3!=NULL)) {
            ______   minuend  _____;
            ______   subtrahend _____;
            minuend = strcmp(arg2,"mat1")==0 ? mat1 :
                      (strcmp(arg2,"mat2")==0 ? mat2 : NULL);
            subtrahend = strcmp(arg3,"mat1")==0 ? mat1 :
                         (strcmp(arg3,"mat2")==0 ? mat2 : NULL);
            if((minuend!=NULL) && (subtrahend!=NULL)) {
                if(mat3!=NULL)
                    free(mat3);
                mat3=sub(minuend,subtrahend);
            }
            else
                printf("Invalid Argument\n");
        }
        else if((strcmp(arg1,"allocv")==0) && (arg2!=NULL)) {
            ______ mat;
            mat=allocv();
            if(strcmp(arg2,"mat1")==0)
                 _______
            if(strcmp(arg2,"mat1")==0)
                 _______
        }
        else if((strcmp(arg1,"debugv")==0) && (arg2!=NULL)) {
            ______ mat;
            if(strcmp(arg2,"mat1")==0)
                 _______
            if(strcmp(arg2,"mat2")==0)
                 _______
            if(strcmp(arg2,"mat3")==0)
                 _______
```

```
            debugv(mat);
        }
        else if((strcmp(arg1,"zeile")==0) && (arg2!=NULL)) {
            int zeilen= (int)strtol(arg2,NULL,0);
            //ggf. konvertieren
            zeile( ___subarray___ );
        }
        else if((strcmp(arg1,"spalte")==0) && (arg2!=NULL)) {
            int spalten=(int)strtol(arg2,NULL,0);
            //ggf. konvertieren
            spalte( ___subarray___ );
        }
#endif
        else if((strcmp(arg1,"dump")==0) && (arg2!=NULL)) {
            if(strcmp(arg2,"mat1")==0)
                dump(mat1,sizeof(mat1),8,DUMP_8);
            if(strcmp(arg2,"mat2")==0)
                dump(mat1,sizeof(mat2),8,DUMP_8);
        }
        else if(strcmp(arg1,"help")==0) {
            printf(help);
        }
        else if(strcmp(arg1,"quit")==0) {
            break;
        }
        else {
            printf("unbekanntes Kommando\n");
        }
    }
    return (int)ret;
}

/
*-----------------------------------------------------------------------------------------------*/
static char *dump_str[]={"8-Bit","16-Bit","32-Bit",
                  "8-Bit","16-Bit","32-Bit",};
//Bei Nutzung dieser Funktion darf der Compiler-Schalter -Fsanitze=address nicht genutzt werden
int dump(void *start, size_t len, int width, DUMP_MODE mode)
{
    void *ptr;
    uintptr_t size;
    switch(mode) {
        case DUMP_8:
        case DUMP_8A:
            size=0x01;
            break;
        case DUMP_16:
        case DUMP_16A:
            size=0x02;
            break;
        case DUMP_32:
        case DUMP_32A:
            size=0x04;
            break;
        default:
            fprintf(stderr,"Illegal Mode\n");
            return -1;
    }
    ptr=(void *)((uintptr_t)start & ~(size-1));
//   printf("Adressbreite: %d Bytes von 0 .... %p\n",sizeof(char *),(uintptr_t)-1);
    printf("--- Dump %10p .. %10p Mode=%s ---",start,start+len-1,dump_str[mode]);
    while(ptr<start+len) {
        printf("\n%0*p:",sizeof(void *)*2+2,ptr);
        for(int lauf=0;lauf<width;lauf++)
            switch(mode) {
                case DUMP_8:
```

```
                case DUMP_8A:
                    printf(" %02x",*(((unsigned char *)ptr)+lauf)&0xFF);
                    break;
                case DUMP_16:
                case DUMP_16A:
                    printf(" %04x",*(((unsigned short *)ptr)+lauf)&0xFFFF);
                    break;
                case DUMP_32:
                case DUMP_32A:
                    printf(" %08x",*(((unsigned int *)ptr)+lauf));
                    break;
            }
        if((mode==DUMP_8A) || (mode==DUMP_16A) || (mode==DUMP_32A)) {
            printf(" - ");
            for(int lauf=0;lauf<(width*size);lauf++)
                switch(mode) {
                    case DUMP_8A:
                    case DUMP_16A:
                    case DUMP_32A:
                        printf("%c",*(((unsigned char *)ptr)+lauf)&0xFF);
                        break;
                    case DUMP_8:
                    case DUMP_16:
                    case DUMP_32:
                        break;
                }
        }
        ptr+=width*size;
    }
    printf("\n");
    return 0;
}
```