Aufgabe (Zeiger)

Montag, 2. Mai 2022 11:41

Übungsaufgaben zu Zeigervariablen

Im nachfolgenden Programm sind Fragestellung formuliert, welche sie bitte nacheinander bearbeiten.

Beachten sie zur Bearbeitung dieser Aufgabe unbedingt folgende Hinweise:

- Sehen sie die Fragen als unabhängige Fragen an, d.h. zuvor getätigte Änderungen bitte vor Beantwortung der nächsten Fragen rückgängig machen.
- **Nutzen sie nicht den Sanitize Compilerschalter**
- Einige Fragen sollten zum Programmabsturz führen (Beim CompilerExplorer durch 'Programm returned: 139' gekennzeichnet). Begründen sie in diesen Fällen, warum es zum Programmabsturz gekommen ist
- Zur Selbstüberprüfung empfiehlt es sich, die dump() Funktion zu nutzen.

```c
#include <stdio.h>
#include <string.h>
#include <stdint.h>   //fuer uintptr_t
//Compilerschalter: keine
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);
typedef  struct  {int  x,y,z;}  xyz_t;
typedef  union   {int  a;short  b;char  c;}  abc_t;
//Bitte als globale Variablen belassen
int     vari1=0x01234567;
short   vars1=0x89ab;
short   vars2=0xcdef;
char    varc1='a';
char    varc2=49;
char    varc3=50;
char    varc4='\a';
int     vari3='a';
char    varc5a[16]="abcdefghijklmno";
xyz_t   varxyz={-2,-3,-4};
abc_t   varabc={.c=3};
unsigned int   *ptri   =(unsigned int   *)0x0001;
unsigned short *ptrs   =(unsigned short *)0x0100;
unsigned char  *ptrc   =(unsigned char  *)0x010000;
xyz_t  *ptrxyz =(xyz_t *)-4;
abc_t  *ptrabc =(abc_t *)-5;
void  func_value(int   vari2) {
         vari2=7;
}
void  func_reference(unsigned int  * vari2) {
        * vari2=7;
}
void  func_ptrptr(unsigned int  ** vari2) {
        * vari2=(unsigned int*)& vari3;
}
void func_return(void) {
        unsigned int   lokale=4711;
        ptri=&lokale;
}
int  main(void)
```

```
{
//1) Nachfolgende Funktion stellt den Speicherbereich rund um die oben
//   definierten globalen Variablen dar. Fügen sie den DUMP-Ausdruck
//   hier in den Kommentarbereich ein und kommentieren sie, welche
//   Inhalte/Bytes zu welchen Variablen gehören
dump(&vari1,(void *)&ptrabc-(void *)&vari1+sizeof(ptrabc),8,DUMP_8A);
//2) Führen sie nachfolgende Befehle im Block aus
//   Erklären sie, für jede einzelne printf Ausgabe, wie die
//   dargestellte Zahl 'zustandekommt'
//   Zur Hilfestellung nutzen sie ergänzend die dump Ausgabe aus der
vorherigen Aufgabe
#if 0
  ptrc= (char  *)& vari1;     fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+1;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+2;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+3;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+4;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+5;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+6;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+7;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+8;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+9;  fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+10; fprintf(stderr,"%02x\n",*ptrc);
  ptrc=((char  *)& vari1)+11; fprintf(stderr,"%02x\n",*ptrc);
#elif 0
  ptrs= (short  *)& vari1;    fprintf(stderr,"%04x\n",*ptrs);
  ptrs=((short  *)& vari1)+2; fprintf(stderr,"%04x\n",*ptrs);
  ptrs=((short  *)& vari1)+3; fprintf(stderr,"%04x\n",*ptrs);
  ptrs=((short  *)& vari1)+4; fprintf(stderr,"%04x\n",*ptrs);
  ptrs=((short  *)& vari1)+5; fprintf(stderr,"%04x\n",*ptrs);
#elif 0
  ptri=& vari1;    fprintf(stderr,"%08x\n",*ptri);
  ptri=& vari1+1;  fprintf(stderr,"%08x\n",*ptri);
  ptri=& vari1+2;  fprintf(stderr,"%08x\n",*ptri);
#endif
//3) Führen sie nachfolgende Zeilen einzeln durch.
//   Begründen sie auf Basis der 'Liste der Prioritäten'
//   https://de.wikibooks.org/wiki/C-Programmierung:
_Liste_der_Operatoren_nach_Priorit%C3%A4t
//   die Abarbeitungsreihenfolge der Operationen.
//   Zur Hilfestellung nutzen sie ergänzend die dump Ausgabe aus der
vorherigen Aufgabe
#if 0
                                    fprintf(stderr,"%x",*ptrc);
ptrc=(unsigned char *)0x100;        fprintf(stderr,"%02x\n",*ptrc);
ptrc=(unsigned char *)&vari1;       fprintf(stderr,"%02x\n",*ptrc);
ptrc=(unsigned char *)&vari1+1;     fprintf(stderr,"%02x\n",*ptrc);
ptrc=(unsigned char *)(&vari1+1);   fprintf(stderr,"%02x\n",*ptrc);
ptrc=(unsigned char *)&vari1;       fprintf(stderr,"%02x\n",*ptrc+1);
ptrc=(unsigned char *)&vari1;       fprintf(stderr,"%02x\n",*(ptrc+1));
ptrc=(unsigned char *)(&vari1-100); fprintf(stderr,"%02x\n",*ptrc);
ptrc=(unsigned char *)(&vari1+100); fprintf(stderr,"%02x\n",*ptrc);
ptrc=(unsigned char *)(&vari1-4096);fprintf(stderr,"%02x\n",*ptrc);
ptrc=(unsigned char *)(&vari1+4096);fprintf(stderr,"%02x\n",*ptrc);
#endif

//4) Lassen sie sich die Inhalte der Variablen vari1..varc4 vor und nach
//   der Ausführung folgender Anweisungen ausgeben
//   Begründen sie die Änderungen, die hier stattfinden
#if 0
  ptrxyz=(xyz_t  *)& vari1;
  ptrxyz->x=0x01234567;
```

```
  ptrxyz->y=0x00020002;
  ptrxyz->z=0x01020403;
#endif
//5) Wie  müssten  die  Variablen   vars2.. varc5a  initialisiert  werden,
//   damit  nachfolgende  Bedingung  erfüllt ist
#if 0
  ptrxyz=(xyz_t*)& vars2;
  if((ptrxyz->x==1)  &&  (ptrxyz->y==-1)  &&  (ptrxyz->z==0x48495051))
        printf("True");
#endif
//6) Erklären sie, wie der dargestellte Speicherabbild 'zustandekommt'
#if 0
  varabc.a=0x11111111;
  varabc.b=0x22222222;
  varabc.c=0x33333333;
  dump(&varabc,4,4,DUMP_8);
#endif
//7) Welche  Auswirkung  hat  der  Aufruf  der  einzelnen  Funktionen  auf
//   die  zuvor  initialisierten globalen Variable/Pointer.
#if 0
  unsigned int vari1=1;
  func_value( vari1);
#elif 0
  unsigned int vari1=1;
  func_reference(& vari1);
#elif 0
  unsigned int vari1=1;
  ptri=& vari1;
  func_reference(ptri);
#elif 0
  unsigned int vari=0;
  ptri=(unsigned int *)& vari1;
  func_ptrptr(&ptri);
#endif
//8) Erklären sie, warum bei den beiden printf Ausgabe unterschiedliche
//   Werte ausgegeben werden, obwohl ptri nur einmal gesetzt wird!
#if 1
volatile unsigned int lok=0;
func_return();
lok=*ptri;
fprintf(stderr,"1.Lauf Value=%u\n",lok);
func_value(1111);
lok=*ptri;
fprintf(stderr,"2.Lauf Value=%u\n",lok);
#endif
//9) varc1 ist eigentlich eine Variable vom Typ Character.
//    Warum entspricht die Adresse dieser Variablen einem 'String'
#if 0
printf("%s\n",&varc1);
#endif
fprintf(stderr,"Main erfolgreich durchgelaufen");
return 0;
}
/
*-------------------------------------------------------------------------
----------------*/
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%10p:",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;
}
```