```
/***************************************************************************/
/*                                                                         */
/*   STACK und RINGLISTE                                                   */
/*   Demonstration eines abstrakten Datenmodelles mit variabler Daten-     */
/*   struktur.                                                             */
/*                                                                         */
/*   (C) by Daniel Becker                    Letzte Änderung: 24.04.99     */
/*                                                                         */
/***************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

/* Vorinitialisiert auf NULL, damit kein Aerger passieren kann... */
/* der Typ des Pointers ist "unsigned char", da ich nur dann wirklich */
/* eine Byteweise addition der Adressen erwarten kann. */

unsigned char *stack=NULL,*stk_pointer,*s_ende;
int s_element_size;

void init_stack(int elements,int size)
{
   if (stack!=NULL) return; /* Stack ist bereits initialisiert! */

   stack=(unsigned char *)malloc(elements*size);
   s_element_size=size;
   stk_pointer=stack;
   s_ende=stack+(elements*size);
}

void clear_stack()
{
   if (stack!=NULL) free(stack); /* Nur initialisierte Stacks freigeben */
   stack=stk_pointer=s_ende=NULL; /* stk_empty funktioniert auch jetzt */
}

int stk_empty()
{
   return(stk_pointer==stack);
}

int stk_full()
{
   return(stk_pointer==s_ende);
}

void push(unsigned char *c)
{
   if (stk_full())
      puts("Stack overflow");
   else
   {
      memcpy(stk_pointer,c,s_element_size); /* MemCopy(dest, Source, len); */
      stk_pointer+=s_element_size;
   }
}

unsigned char *pop()
{
   if (stk_empty())
      puts("Stack underflow");
   else
   {
      stk_pointer-=s_element_size;
      return(stk_pointer);
   }
}

void smart_pop(unsigned char *c)
{
   if (stk_empty())
      puts("Stack underflow");
```

```
   else
   {
      stk_pointer-=s_element_size;
      memcpy(c,stk_pointer,s_element_size);
   }
}

unsigned char *top_of_stack()
{
   return(stk_pointer-s_element_size);
}

/*****************************************************************************/

unsigned char *ringlist=NULL,*pushlist,*poplist,*l_ende;
int l_element_size;

void init_list(int elements,int size)
{
   if (ringlist!=NULL) return; /* Liste ist bereits initialisiert! */

   ringlist=(unsigned char *)malloc(elements*size);
   l_element_size=size;
   pushlist=poplist=ringlist;
   l_ende=ringlist+(elements*size);
}

void clear_list()
{
   if (ringlist!=NULL) free(ringlist); /* Nur initialisierte Listen freigeben */
   ringlist=NULL;
}

unsigned char *addone(unsigned char *p)
{
   p+=l_element_size;
   if (p>l_ende) p=ringlist;
   return(p);
}

int list_empty()
{
   return(pushlist==poplist);
}

int list_full()
{
   return(addone(pushlist)==poplist); /* Auf ein Element ran=Voll! */
}

void enqueue(unsigned char *c)
{
   if (list_full())
      puts("List overflow");
   else
   {
      memcpy(pushlist,c,l_element_size);
      pushlist=addone(pushlist);
   }
}

unsigned char *dequeue()
{
   if (list_empty())
      puts("List underflow");
   else
   {
      unsigned char *c; /* kleiner Helfer */

      c=poplist;
      poplist=addone(poplist);
      return(c);
   }
```

```
}

unsigned char *actual_list_element()
{
   return(poplist);
}

/*****************************************************************************/
/*****************************************************************************/
/*             Testanwendung hierzu                                          */

void stk_demo()
{
   int i;
   struct fun
   {
      int i;
      char str[80];
   } test,*pop_ergebnis;
   /* Struktur zur demonstration der Vielseitigkeit */

   printf("Demonstration der Stackstruktur.\n\nAls erstes reine Zahlen (mit Stack overflow
   und underflow)\n");
   init_stack(10,sizeof(int)); /* Ein Stack aus Integer-Werten */

   /* Einfach ein paar Werte drauf */
   i=0;
   while (!stk_full())
   {
      push((unsigned char *)&i);
      i++;
   }
   /* Fehler erzeugen: */
   push((unsigned char *)&i);

   /* und wieder runter... */
   while (!stk_empty()) printf("%d, ",*(unsigned int *)(pop()));

   /* und Fehler demonstrieren: */
   pop();
   /* keine Variable für das Ergebnis notwendig -> Vergesssen wir's */

   /* Stack wieder aufräumen. */
   clear_stack();

   printf("\nJetzt Elemente einer Struktur:\n");

   init_stack(10,sizeof(struct fun));

   /* Jetzt eine struktur. Wieder ein Int als Zaehler */
   test.i=0;
   while (!stk_full())
   {
      sprintf(test.str,"STRNR %d",test.i);
      /* sprintf wie printf, nur Ausgabe in einen String. */

      push((unsigned char *)&test);
      /* Typecast notwendig und nicht schädlich! */

      test.i++;
   }

   /* Und wieder alles runter: */
   while (!stk_empty())
   {
      pop_ergebnis=(struct fun *)pop();
      /* Wieder ein Typecast! */

      printf("%d[%s], ",pop_ergebnis->i,pop_ergebnis->str);
      /* Zugriff auf Strukturmitglieder mit Pfeil, wenn */
      /* Strukturname nur ein Pointer ist. Sonst mit Punkt */
   }
```

```
   /* Und wieder aufraeumen... */
   clear_stack();

   printf("\n");
}

void liste_demo()
{
   int i;
   struct fun
   {
      int i;
      char str[80];
   } test,*pop_ergebnis;
   /* Struktur zur demonstration der Vielseitigkeit */

   printf("\n\nDemonstration der Listenstruktur\n\nAls erstes reine Zahlen (mit List overflow
   und underflow):\n");
   init_list(10,sizeof(int)); /* Eine Liste aus Integer-Werten */

   /* Einfach ein paar Werte rein */
   i=0;
   while (!list_full())
   {
      enqueue((unsigned char *)&i);
      i++;
   }
   /* Fehler erzeugen: */
   enqueue((unsigned char *)&i);

   /* und wieder raus... */
   while (!list_empty()) printf("%d, ",*(unsigned int *)(dequeue()));

   /* und Fehler demonstrieren: */
   dequeue();
   /* keine Variable für das Ergebnis notwendig -> Vergesssen wir's */

   /* Liste wieder aufräumen. */
   clear_list();

   printf("\nJetzt Elemente einer Struktur:\n");

   init_list(10,sizeof(struct fun));

   /* Jetzt eine struktur. Wieder ein Int als Zaehler */
   test.i=0;
   while (!list_full())
   {
      sprintf(test.str,"STRNR %d",test.i);
      /* sprintf wie printf, nur Ausgabe in einen String. */

      enqueue((unsigned char *)&test);
      /* Typecast notwendig und nicht schädlich! */

      test.i++;
   }

   /* Und wieder alles runter: */
   while (!list_empty())
   {
      pop_ergebnis=(struct fun *)dequeue();
      /* Wieder ein Typecast! */

      printf("%d[%s], ",pop_ergebnis->i,pop_ergebnis->str);
      /* Zugriff auf Strukturmitglieder mit Pfeil, wenn */
      /* Strukturname nur ein Pointer ist. Sonst mit Punkt */
   }
   /* Und wieder aufraeumen... */
   clear_list();

   printf("\n\nFertig\n");
}
```

```
void main()
{
   clrscr(); /* Bildschirm erst mal leer */

   stk_demo();
   liste_demo();
}SUB
```