ZINSEN.C Übung 1, Aufgabe 2

```
#include <stdio.h>
#include <math.h>

int main()
{
  double einzahlung= 500; // 500 Euro
  double guthaben;
  double zinssatz;

  // Nach Einzahlung
  guthaben = einzahlung;

  // Nach einem Jahr mit dem Zinssatz 0.5 %
  zinssatz = 0.5;
  guthaben = guthaben * (1 + zinssatz / 100 ) ;
  printf("Guthaben nach 1. Jahr : %lf \n", guthaben);


  getchar();
  return 0;
}
```

GGT größter gemeinsamer Teiler //Übung 1, Aufgabe 1

```
#include <stdio.h>

int main()
{
int a, b, temp, ggt;
double q;
printf("Berechnung ggt ...\n");

// Eingabe a,b
printf("a:");
scanf("%d",&a);
printf("b:");
scanf("%d",&b);

// fuehre solange aus wie a ungleich b
while(a!=b)  // entspricht Regel 2
{
 if (b>a) // Regel 1
 {
 temp = a;  // tausch a,b
 a = b;
 b = temp;
 }
 a= a-b; // Regel 3
}
ggt= a;

printf("ggt ergibt %d\n", ggt);

getchar();
getchar();
return 0;
}
```

alternierende_reihe.c

```
#include <stdio.h>

int main()
{
  double vz=+1.0;
    double sum = 0;
    double summand;
    int z;
    //char ende=0;

    z=1;
    while( 1 ) // auch while (!ende) möglich
    {
       summand = 1.0/z; // 1/z würde bei z>1 immer Null ergeben
       if ( summand < 0.000001)
          break;
          //ende=1;   // wäre anstelle break auch möglich
       sum = sum + vz* summand;

       if (vz>0)
          vz=-1.0;
       else
          vz=+1.0;
       z=z+1;
    }

    printf("Summe nach %d Durchlaeufen: %lf\n", z, sum);

    getchar();

    return 0;
}
```

FIBONACCI-ZAHLEN

```
#include <stdio.h>

// Die Funktion fibo() ist eine rekursive Funktion
int fibo( int n)
{
  if (n<2)
     return 1;
  else
     return fibo(n-2)+fibo(n-1);
}


int main()
{
  int x=5;
  int i, f;

  f = fibo(x);
  printf("fibo(%d) ergibt %d \n", x,f);

  // wir lassen uns die Werte fibo(0) bis fibo(40) ausgeben
  for (i=0;i<40;i++)
    printf("fibo(%d) ergibt %d \n", i,fibo(i));

  getchar();
  return 0;
}
```

BEISPIEL_DATEI_ERWEITERT.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100
struct laeufer {
  char vname[20],name[20];
  int h,m,s;
};

struct laeufer_listelem {
  char vname[20],name[20];
  int h,m,s;
  struct laeufer_listelem *next;
};

int main()
{
   FILE *f;
   int n;
   int i,anz_zeilen = 0;
   struct laeufer l;

   struct laeufer alle[MAX];
   struct laeufer *dynarray;
   struct laeufer_listelem *anker, *neu, *ende;

   f=fopen("C:\\TEMP\\allelaeufer.txt","rt");

   if (f==NULL)
   {
     printf("Datei konnte nicht geoeffnet werden.\n");
     return -1;
   }

   // Einlesen und Ausgeben auf Konsole
   while (!feof(f))
   {
      n= fscanf(f,"%s %s %d:%d:%d",l.vname,l.name,&l.h,&l.m,&l.s);
      if (n==5)
      {
        printf("n=%d, Gelesen: %s %s  %02d:%02d:%02d\n", n, l.vname,l.name,l.h,l.m,l.s);
        // hier die gelesenen Daten verarbeiten
```

```
     anz_zeilen ++;
   }
   else
   {
    printf("n=%d, Gelesen: %s %s  %02d:%02d:%02d\n", n, l.vname,l.name,l.h,l.m,l.s);
    // hier ggf. Endebehandlung durchführen
   }
}

printf("Es wurden %d vollstaendige Zeilen gelesen\n",anz_zeilen);

// Einlesen auf Array fester Größe
rewind(f);
i=0;
while (!feof(f))
{
   n= fscanf(f,"%s %s %d:%d:%d",l.vname,l.name,&l.h,&l.m,&l.s);
   if (n==5)
   {
    alle[i]=l;
    i++;
   }
   else
   {
    printf("Ende erreicht\n");
   }
}

// Einlesen auf dynamisches Array
dynarray = malloc(anz_zeilen*sizeof(struct laeufer));
rewind(f);
i=0;
while (!feof(f))
{
   n= fscanf(f,"%s %s %d:%d:%d",l.vname,l.name,&l.h,&l.m,&l.s);
   if (n==5)
   {
      dynarray[i]=l;
      i++;
   }
   else
   {
      printf("Ende erreicht\n");
   }
}
// Einlesen auf Liste
```

```
   anker=NULL; ende=NULL;
   rewind(f);

   while (!feof(f))
   {
      n= fscanf(f,"%s %s %d:%d:%d",l.vname,l.name,&l.h,&l.m,&l.s);
      if (n==5)
      {
         neu =malloc(sizeof(struct laeufer_listelem));
         strcpy(neu->vname,l.vname);
         strcpy(neu->name,l.name);
         neu->h=l.h;
         neu->m=l.m;
         neu->s=l.s;
         neu->next=NULL;
         if (anker==NULL)
         {    anker=neu;
              ende = neu;
         }
         else
         {
            ende->next=neu;
            ende=neu;
         }

      }
      else
      {
         printf("Ende erreicht\n");
      }
   }

   fclose(f);
   return 0;
}
```

BEISPIEL_BINÄRDATEI.C

```
// Daten auf Binärdatei schreiben und danach wieder lesen
#include <stdio.h>
#include <string.h>

#define ERZEUGEN_UND_SCHREIBEN
//#define LESEN_UND_AUSGEBEN

typedef enum {grad_celsius, hektopascal} einheit_t;

typedef struct{
  int h,m,s;
} zeit_t;

typedef struct {
  zeit_t zeit;
  char ort[20];
  double wert;
  einheit_t e;
} messung_t;

void print_messung(messung_t m)
{
  char einh[20]="";
  if (m.e==hektopascal)
    strcpy(einh,"Hektopascal");
  if (m.e==grad_celsius)
    strcpy(einh,"Grad Celsius");

  printf("%02d:%02d:%02d, %20s: %lf %s\n",
        m.zeit.h, m.zeit.m, m.zeit.s, m.ort, m.wert, einh);
}

int main()
{
  FILE *f;
  messung_t m1,m2;

#ifdef ERZEUGEN_UND_SCHREIBEN
  m1.zeit.h=12;
  m1.zeit.m=0;
  m1.zeit.s=0;
  strcpy(m1.ort,"Dresden-Klotzsche");
  m1.wert=987.5;
  m1.e = hektopascal;
```

```
  m2=m1;
  m1.wert=12.8;
  m1.e = grad_celsius;

  f=fopen("C:\\Temp\\binaerdatei.bin", "wb");
  fwrite(&m1,sizeof(m1),1,f);
  fwrite(&m2,sizeof(m2),1,f);
  fclose(f);

  printf("Geschrieben:\n");
  print_messung(m1);
  print_messung(m2);

#endif
#ifdef LESEN_UND_AUSGEBEN
  f=fopen("C:\\Temp\\binaerdatei.bin", "rb");
  fread(&m1,sizeof(m1),1,f);
  fread(&m2,sizeof(m2),1,f);
  fclose(f);

  printf("Gelesen:\n");
  print_messung(m1);
  print_messung(m2);
#endif
  getchar();
  return 0;
}
```

DYNAMISCHES_ARRAY.C

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <memory.h>
#define STRLEN 100
#define N_START 3
#define N_DELTA 2


// Struktur für Daten
typedef struct {
 int id;
 char name[STRLEN];
} element_t;

// Struktur zur Verwaltung des dynamischen Speichers
// Zeiger und Information über genutzte und allokierte Elemente zusammengefasst
typedef struct {
 element_t *array_ptr;
 int n_elements;
 int n_allocated;
} dynarray_handle;

char pruefe_eingabe(char *string)
{
 unsigned int i;
 while(string[0]==' ')
  strcpy(string,&string[1]);
 for (i=0;i<strlen(string);i++)
  string[i]=toupper(string[i]);
 return string[0];
}
void anzeigen(dynarray_handle dh)
{
 int i;
 printf("Dynamisches Array, Elemente: %d genutzt, %d allokiert \n",dh.n_elements, dh.n_allocated);

 for (i=0;i<dh.n_elements;i++)
 printf("ID: %d \t Name: %s\n", dh.array_ptr[i].id, dh.array_ptr[i].name);
 printf("\n");
}

int einfuegen(dynarray_handle *dh, int id, char *name)
```

```
{
  if (dh->n_elements==0)
  { dh->array_ptr = (element_t*) malloc(sizeof(element_t)* N_START);
    if (dh->array_ptr==NULL) { return 0; }
    dh->n_allocated = N_START;
  }
  if (dh->n_elements+1> dh->n_allocated)
  { dh->array_ptr = (element_t*) realloc(dh->array_ptr, sizeof(element_t) *
                                  (dh->n_allocated + N_DELTA));
    if (dh->array_ptr==NULL) { return 0; }
    dh->n_allocated = dh->n_allocated + N_DELTA;
  }
  dh->array_ptr[dh->n_elements].id = id;
  strcpy(dh->array_ptr[dh->n_elements].name, name);
  dh->n_elements = dh->n_elements+1;
  return 1;
}
int loeschen(dynarray_handle *dh, int id)
{
  int i, p=-1;

  for(i=0; i<dh->n_elements; i++)
    if (dh->array_ptr[i].id == id)
    { p=i;
      break;
    }
  if (p>=0) // wurde Loeschposition gefunden?
  {
    for (i=p+1;i<dh->n_elements; i++)
      dh->array_ptr[i-1] = dh->array_ptr[i];
    dh->n_elements = dh->n_elements - 1;

    if (dh->n_elements < dh->n_allocated - N_DELTA)
    {
      dh->array_ptr = (element_t*) realloc(dh->array_ptr, sizeof(element_t) *
                                    (dh->n_allocated - N_DELTA));
      dh->n_allocated = dh->n_allocated - N_DELTA;
    }
    return 1;
  }
  return 0;
}
int main()
{
  char eingabestring[STRLEN], name[STRLEN];
  char auswahl;
```

```
 int id, rc;
 char ende = 0;

 dynarray_handle dh = {NULL, 0, 0};

 printf("Demonstration eines dynamischen Arrays\n");
 do {
  printf("Funktionsauswahl [E]infuegen [L]oeschen [A]nzeigen [Q]uit :");
  scanf("%s",eingabestring);
  auswahl = pruefe_eingabe(eingabestring);

  switch(auswahl)
  {
    case 'A':
          anzeigen(dh);
          break;
    case 'E':
          printf("Eingabe id:"); scanf("%d",&id);
          printf("Eingabe name:"); scanf("%s",name);
          rc = einfuegen(&dh, id, name);
          if (rc==1)
            printf("Neues Element uebernommen: Elemente: %d genutzt, %d allokiert \n",
                dh.n_elements, dh.n_allocated);
          else
            printf("Fehler bein Einfuegen: Elemente: %d genutzt, %d allokiert \n",
                dh.n_elements, dh.n_allocated);
          break;
    case 'L':
          printf("Eingabe id:"); scanf("%d",&id);
          rc = loeschen(&dh, id);
          if (rc==1)
            printf("Element geloescht: Elemente: %d genutzt, %d allokiert \n",
                dh.n_elements, dh.n_allocated);
          else
            printf("Element wurde nicht gefunden\n");
          break;
    case 'Q':
          ende = 1;
          break;
  }
 } while(!ende);
 printf("\nEnde\n");
 return 0;
}
```

LINEARELISTE.C

```
// Beispiel für eine Verkettete Liste
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define STRLEN 80

struct listelem
{ float t;
  char position[STRLEN]; // anwendungsspez. Elemente
  struct listelem *next;
};

typedef  struct listelem  list_elem_t;

// ---------- Begin list operations -------

list_elem_t *create(list_elem_t x)
{  list_elem_t *neu;
   neu= (list_elem_t*) malloc(sizeof x);    //Reservieren von Speicherplatz
   *neu=x;                              //Belegung des Speichers
   return neu;
}

void insert(list_elem_t *pos, list_elem_t *neu)
{  // pos zeigt auf das Listenelement, hinter dem das
   // Listenelement neu eingekettet werden soll
   neu->next=pos->next;
   pos->next = neu;
}

void dequeue(list_elem_t *pos)
{  // pos zeigt auf Element vor dem auszukettenden Element
   list_elem_t *h;
   h=pos->next;
   pos->next=(pos->next)->next;
   free(h);
}
// -----------   end list operations ------------------

void Insert(list_elem_t **anker, list_elem_t insertelem)
{
   list_elem_t *newelem = create (insertelem);
```

```
  // insert after last entry that contains a smaller t
  list_elem_t *ptr = *anker;
  list_elem_t *pre = NULL;

  if (ptr==NULL)
  { // start from an empty list
    *anker = newelem;
    (*anker)->next = NULL;
  }
  else
  {
    while (ptr)
    {
      if (ptr->t < newelem->t)
      { pre = ptr;
        ptr = ptr -> next;
      }
      else
      { if (pre)
          insert (pre, newelem);
        else
        { newelem->next= *anker;
          *anker = newelem;
        }
       return;
      }
    }
    // travesersed without insertion -> add at the end
    insert(pre, newelem);
  }
}

void Delete(list_elem_t **anker, char *delposition)
{
  list_elem_t *ptr = *anker;
  list_elem_t *pre = NULL;

  while (ptr)
  {
    if (strcmp(ptr->position, delposition)) // dann ungleich
    { pre = ptr;
      ptr = ptr -> next;
    }
    else // dann muss Element gelöscht werden
    { if (pre)
```

```
        dequeue(pre);
      else
      { // Sonderfall: erstes Element wird geloescht
        list_elem_t *tmp;
        tmp = (*anker)->next;
        free(*anker);
        *anker = tmp;
      }
      return;
    }
  }
}

void Listing(list_elem_t *anker)
{
  list_elem_t *ptr = anker;
  while (ptr)
  {
    printf("list elem at adr=%p [%6.2f, %s] next=%p \n",
          ptr, ptr->t, ptr->position, ptr->next);
    ptr = ptr->next;
  }
}

int main()
{
  list_elem_t *anker = NULL;

  char cmd[STRLEN];
  do {
    printf("Command i(nsert) d(elete) l(ist) q(uit)");
    fgets(cmd, STRLEN, stdin);

    if ( !strncmp(cmd,"insert",7) || !strncmp(cmd,"Insert",7) ||
         !strncmp(cmd,"i",1) ||  !strncmp(cmd,"I",1) )
    {
      list_elem_t insertelem;
      char *nlc, tstring[STRLEN];

      printf("Insert element:\n");
      printf("Position:"); fgets(insertelem.position, STRLEN, stdin);
      nlc=strchr(insertelem.position,'\n');
      if (nlc) *nlc='\0';
      printf("Temperature:"); fgets(tstring, STRLEN, stdin);
      sscanf(tstring, "%f",&insertelem.t);
      Insert(&anker, insertelem);
```

```
      continue;
    }
    if ( !strncmp(cmd,"delete",6) || !strncmp(cmd,"Delete",6) ||
         !strncmp(cmd,"d",1) ||  !strncmp(cmd,"D",1) )
    {
      char del_position[STRLEN];
      char *nlc;

      printf("Delete element:\n");
      printf("Position:"); fgets(del_position, STRLEN, stdin);
      nlc=strchr(del_position,'\n');
      if (nlc) *nlc='\0';
      Delete (&anker, del_position);

      continue;
    }

    if ( !strncmp(cmd,"list",4) || !strncmp(cmd,"List",4) ||
         !strncmp(cmd,"l",1) ||  !strncmp(cmd,"L",1) )
    {
      printf("Listing of elements:\n");

      Listing(anker);

      continue;
    }
    if ( !strncmp(cmd,"quit",4) || !strncmp(cmd,"Quit",4) ||
         !strncmp(cmd,"q",1) ||  !strncmp(cmd,"Q",1) )
    { printf("Program end\n");
      break;
    }
  } while (1);
  return 0;
}
```

ÜBUNG6_STACT_DEMO.C

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <malloc.h>

typedef struct {
  int id;
  char name[20];
} element_t;

typedef struct {
  element_t *array;
  int n_alloc;
  int n_elem;
} stack_t;

int stack_init(stack_t *s, int n)
{
  s->array = (element_t*) malloc(n*sizeof(element_t));
  s->n_alloc=n;
  s->n_elem=0;
  if (s->array!=NULL) return 1;
  else return 0;
}

int stack_push(stack_t *s, element_t e)
{
  if (s->n_elem == s->n_alloc)
    return 0;
  else
  {
    s->array[s->n_elem] = e;
    s->n_elem++;
    return 1;
  }
}

int stack_pop(stack_t *s, element_t *e)
{
  if (s->n_elem == 0)
    return 0;
  else
  {
    *e = s->array[s->n_elem-1];
    s->n_elem--;
```

```
    return 1;
  }
}

int main()
{
  stack_t s, ss;
  int i,j;
  element_t e_in;
  element_t e_out;

  stack_init(&s,100);

  printf("Stack-Elemente zufuegen:\n");
  for (i=0;i<10;i++)
  { e_in.id=i;
    sprintf(e_in.name,"name%d-ABC",i);

    printf("PUSH: %d %s\n", e_in.id, e_in.name);

    stack_push(&s, e_in);
  }

  printf("Stack-Elemente auslesen:\n");
  while (stack_pop(&s, &e_out))
  {
    printf("POP:  %d %s\n", e_out.id, e_out.name);
  }

  getchar();
  return 0;
}
```

ÜBUNG6_WARTESCHLANGE_DEMO.C

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <malloc.h>

typedef struct {
  int id;
  char name[20];
} element_t;

typedef struct {
  element_t *array;
  int n_alloc;
  int start, end, n_elem;
} queue_t;

int queue_init(queue_t *q, int n)
{
  q->array = (element_t*) malloc(n*sizeof(element_t));
  q->n_alloc=n;
  q->n_elem=0;
  q->start=0;
  q->end=0;
  if (q->array!=NULL) return 1;
  else return 0;
}

int queue_in(queue_t *q, element_t e)
{
  if (q->n_elem == q->n_alloc)
    return 0;
  else
  {
    q->array[q->end] = e;
    q->n_elem++;
    q->end = (q->end+1)%q->n_alloc;
    return 1;
  }
}

int queue_out(queue_t *q, element_t *e)
{
  if (q->n_elem == 0)
    return 0;
  else
```

```
  {
    *e = q->array[q->start];
    q->n_elem--;
    q->start = (q->start+1)%q->n_alloc;
    return 1;
  }
}
/* ---------------------------------------- */

int main()
{
  int i,j;
  queue_t q;
  element_t e_in;
  element_t e_out;

  queue_init(&q,20);

  for(j=0;j<3;j++)
  {
    printf("Einfuegen in Warteschlange:\n");
    for(i=0;i<10;i++)
    {
      e_in.id=i; sprintf(e_in.name,"name%d-ABC",i);
      printf("queue_in: %d %s\n", e_in.id, e_in.name);
      queue_in(&q, e_in);
    }

    printf("Auslesen aus Warteschlange:\n");
    while (queue_out(&q, &e_out))
    {
      printf("queue_out: %d %s\n", e_out.id, e_out.name);
    }
  }
  getchar();
  return 0;
}
```

ÜBUNG7_DYNAMISCHE_MATRIX.C

```
// Beispiel zur Erzeugung und Handhabung
// einer n x n-Matrix dynamischer Groesse,
// n wird erst zu Laufzeit bekannt

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

int main()
{
 int n,z,s;
 double **m;
 double *matrixflat;
 printf("n:");
 scanf("%d",&n);

 // Allokieren
 m = (double**) malloc(n*sizeof(double*));
 for (z=0;z<n;z++)
  m[z] = (double*) malloc(n*sizeof(double));

 // Belegen
 for (z=0;z<n;z++)
  for (s=0;s<n;s++)
    m[z][s]=(double)(z+s);

 //Ausgeben
 for (z=0;z<n;z++)
 { for(s=0;s<n;s++)
    printf(" %6.1lf ",m[z][s]);
  printf("\n");
 }

 // Freigeben
 for (z=0;z<n;z++)
  free(m[z]);
 free(m);

 getchar();
 getchar();
 return 0;
}
```

ÜBUNG7_BINÄRBAUM.C

```
// Prog.-1, Uebung 7
// Demonstration eines Binaerbaums
// zur Speicherung von Artikeldaten

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

struct artikel{
  int anr;      // Artikelnummer
  char bez[80]; // Bezeichnung
  double preis;
  int bestand;  // Anzahl Artikel im Lager
};
struct baum_element {
  struct artikel a;
  struct baum_element *li, *re;
};

struct artikel suche(struct baum_element *wrzl, int anr_gesucht)
{
  struct artikel leer={0,"",0.0,0};

  if (wrzl==NULL)
    return leer;

  if (wrzl->a.anr==anr_gesucht)
    return wrzl->a;
  if (anr_gesucht<wrzl->a.anr)
    return suche(wrzl->li, anr_gesucht);
  else
    return suche(wrzl->re, anr_gesucht);
}

double gesamtwert(struct baum_element *wrzl)
{
  double g;
  if (wrzl==NULL)
    return 0;

  g = wrzl->a.preis*wrzl->a.bestand;
  g = g + gesamtwert(wrzl->li);
  g = g + gesamtwert(wrzl->re);
```

```
  return g;
}

void init_artikel(struct artikel *a, int anr, char *bez, double preis, int bestand)
{
  a->anr = anr;
  strcpy(a->bez,bez);
  a->preis = preis;
  a->bestand=bestand;
}

int main()
{
  struct artikel a;
  double gw;

  // Baum aufbauen
  struct baum_element *w = ( struct baum_element*) malloc(sizeof(struct baum_element));
  init_artikel(&w->a,2342, "Zigaretten Schwarzer Krauser", 3.95, 100);
  w->li = ( struct baum_element*) malloc(sizeof(struct baum_element));
  init_artikel(&w->li->a,1242, "0.75 Liter Rum Verschnitt ", 12.99, 10);
  w->li->li=NULL;
  w->li->re=NULL;
  w->re = ( struct baum_element*) malloc(sizeof(struct baum_element));
  init_artikel(&w->re->a,3212, "Schlagsahne 100g Becher", 0.79, 56);
  w->re->li=NULL;
  w->re->re=NULL;

  // Binaerbaum kann mit weiteren Knoten ausgestattet werden


  // hier kann die Such-Artikelnummer geaendert werden. Wird der Artikel gefunden?
  a = suche(w, 1242);
  printf("Gesucht und gefunden: %d, %s, %lf, %d\n", a.anr, a.bez, a.preis, a.bestand  );

  gw=gesamtwert(w);
  printf("Gesamtwert: %lf\n",gw);

  getchar();
  return 0;
}
```