Technology
Arts Sciences
TH Köln

ALGORITHMEN UND PROGRAMMIERUNG I
WS 2020/2021
Prof. Dr. Frank Victor

Praktikum 5 Abgabe bis 17.01.2021

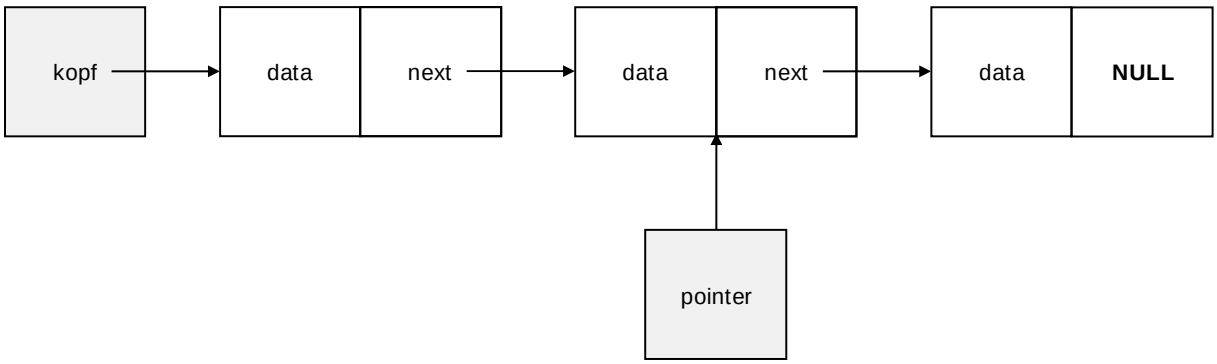
Name, Vorname: Özkurt, Cihat
Mat.-Nr.: 11148632
GMID: inf2323
Datum: 17.01.2021
Betreuer im Praktikum: Gross, Julian

Aufgabe 1: Programmieren in C

Haben Sie sich auch schon einmal über die Arrays geärgert? Darüber, dass die Länge fest vorgegeben sein muss? Dass es schwierig ist, ein Element an eine beliebige Stelle in einem Array einzufügen? Dass, wenn wir Elemente löschen, Lücken in den Arrays entstehen?

All diesen Ärger kann man vermeiden, wenn man **dynamische Datenstrukturen** benutzt. Eine, die wir in dieser Aufgabe behandeln wollen, ist die (einfach) **verkettete Liste** (engl. **Linked List**). Allerdings braucht man dazu Pointer – was gerade für Anfänger auch ein Hindernis sein und Frust bedeuten kann.

Eine verkettete Liste besteht aus Knoten. Jeder Knoten hat einen Inhalt (**data**) und einen Pointer auf seinen Nachfolger (**next**).



Es gibt immer einen Pointer auf den Anfang der Liste. Diesen nennt man **Kopf** (Head). Das Ende der Liste wird durch den Eintrag **NULL** für next im letzten Element beschrieben. Dieser Pointer next zeigt eben auf keinen Speicherplatz.

Will man Algorithmen für Linked Lists definieren (wie Einfügen oder Löschen), so braucht man häufig einen oder mehrere Hilfszeiger (pointer).

Man kann sich nun leicht vorstellen, wie das Ganze funktioniert. Wenn ich beispielsweise ein Element löschen will, so kann ich das tun, indem ich einfach den next-Zeiger des Vorgängers umhänge und den Speicherplatz des Elements freigebe.

Bitte schauen Sie sich das folgende Programm an, **laden Sie die Datei linkedList.c auf die advm1** und führen Sie das Programm aus. Versuchen Sie zu verstehen, was hier passiert.

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

typedef struct Knoten_ {
        int data;
        struct Knoten_ *next;
} Knoten;

Knoten *kopf = NULL;
int anzahl = 0;


void einfuegenAmAnfang(int x) {
   Knoten *t;

   t = (Knoten *)malloc(sizeof(Knoten));
   anzahl++;

   if (kopf == NULL) {
      kopf = t;
      kopf->data = x;
      kopf->next = NULL;
   }
   else {
      t->data = x;
      t->next = kopf;
      kopf = t;
   }
}

void einfuegenAmEnde(int x) {
   Knoten *t, *temp;

   t = (Knoten*)malloc(sizeof(Knoten));
   anzahl++;

   if (kopf == NULL) {
      kopf = t;
      kopf->data = x;
      kopf->next = NULL;
   }
   else {
      temp = kopf;

      while (temp->next != NULL)
         temp = temp->next;

      temp->next = t;
      t->data    = x;
      t->next    = NULL;
   }
}

void ausgeben() {
   Knoten *t;

   t = kopf;

   if (t == NULL) {
      printf("Die Linked List ist leer.\n");
   }
   else {
      printf("Die Linked List hat %d Elemente.\n", anzahl);

```

```
65       while (t->next != NULL) {
66          printf("%d ", t->data);
67       t = t->next; 68
         }
69       printf("%d\n", t->data);
70    }
71 }
72
73 void loeschenAmAnfang() {
74    Knoten *t;
75    int n;
76
77    if (kopf == NULL) {
78    printf("Die Linked List ist schon leer.\n"); 79
      }
80    else {
81       n = kopf->data;
82       t = kopf->next;
83       free(kopf);
84       kopf = t;
85       anzahl--;
86    printf("%d wurde vom Anfang geloescht.\n", n); 87
      }
88 }
89
90 void loeschenAmEnde() {
91    Knoten *t, *u;
92    int n;
93
94    if (kopf == NULL) {
95    printf("Die Linked List ist schon leer.\n"); 96
      }
97    else {
98       anzahl--;
99
100      if (kopf->next == NULL) {
101         n = kopf->data;
102         free(kopf);
103         kopf = NULL;
104         printf("%d wurde vom Ende geloescht.\n", n);
105      }
106      else {
107        t = kopf;
108
109        while (t->next != NULL) {
110           u = t;
111           t = t->next;
112        }
113
114        n = t->data;
115        u->next = NULL;
116        free(t);
117        printf("%d wurde vom Ende geloescht.\n", n);
118      }
119   }
120 }
121
122 int main() {
123    int auswahl, data;
124    int weiter = 1;
125
126    while (weiter) {
127       printf("\nWas wollen Sie mit der Linked List tun?\n\n");
128       printf("1. Einfuegen am Anfang\n");
129       printf("2. Einfuegen am Ende\n");
```

```
130         printf("3. Ausgabe der Linked List\n");
131         printf("4. Loeschen am Anfang\n");
132         printf("5. Loeschen am Ende\n");
133         printf("6. Ende\n\n");
134         printf("Auswahl: ");
135         scanf("%d", &auswahl);
136         printf("\n");
137
138         switch (auswahl) {
139            case 1:
140               printf("Wert eingeben: ");
141               scanf("%d", &data);
142               einfuegenAmAnfang(data);
143               break;
144            case 2:
145               printf("Wert eingeben: ");
146               scanf("%d", &data);
147               einfuegenAmEnde(data);
148               break;
149            case 3:
150               ausgeben();
151               break;
152            case 4:
153               loeschenAmAnfang();
154               break;
155            case 5:
156               loeschenAmEnde();
157               break;
158            case 6:
159               weiter = 0;
160               break;
161            default:
162               printf("Bitte einen sinnvollen Wert eingeben. \n");
163         }
164      }
165      return 0;
166 }
```

Beanworten Sie die folgenden Fragen schriftlich.

Frage 1: Wie funktioniert `malloc` in Zeile 16?

Lösung: Die "malloc" Funktion in C wird verwendet, um einen einzelnen großen Speicherblock mit der angegebenen Größe dynamisch zuzuweisen.

Frage 2: Was macht `free` in Zeile 83? Was passiert, wenn man es weglässt?

Lösung: Die Funktion free () wird aufgerufen, um Speicher freizugeben. Indem Sie Speicher in Ihrem Programm freigeben, stellen Sie mehr für die spätere Verwendung zur Verfügung. Wenn man es weglässt, tritt möglicherweise ein Speicherfehler auf.

Frage 3: Was passiert in Zeile 9? Was ist ein `NULL`-Pointer?

Lösung: Das ist ein Pointer. Seine Typ ist Knot.er hat 2 Variable.Er zeigt Null Pointer weil es keine wert drin gibt.

Frage 4: Was sind die entscheidenden Vorteile einer Linked List im Vergleich zu einer Realisierung einer Liste als Array?

Lösung: Die Größe der Arrays ist festgelegt: Wir müssen also die Obergrenze für die Anzahl der Elemente im Voraus kennen. Außerdem ist der zugewiesene Speicher im Allgemeinen unabhängig von der Verwendung gleich der Obergrenze, und in der Praxis wird die Obergrenze selten erreicht.
Das Einfügen eines neuen Elements in ein "Array" von Elementen ist teuer, da Platz für die neuen Elemente geschaffen werden muss und vorhandene Räume verschoben werden müssen, um Raum zu schaffen.

Aufgabe 2: Programmieren in C

Erweitern Sie das Programm um eine Funktion **`void loescheKnoten(int x).`** Hier wird der Knoten mit dem Wert **`x`** gesucht und aus der Liste gelöscht. Es soll entweder ausgegeben werden, dass **`x`** nicht vorkommt oder aber „Knoten mit dem Wert **`x`** wurde gelöscht".

Hinweis: Sie benötigen hier neben dem Kopf 2 weitere Pointer, einen Hilfszeiger – nennen wir in **`t`**, mit dem Sie die Liste durchlaufen, und einen, mit dem Sie sich den Vorgängerknoten merken, nennen wir in **`prev`** (für previous, vorangegangen).

Algorithmus:

(1) Wenn der Kopfzeiger schon auf den richtigen Wert zeigt, dann sind Sie fertig. Sie müssen dann nur den Hilfszeiger **`t`** freigeben.
(2) Andernfalls müssen Sie die Liste durchlaufen. Merken Sie sich im Zeiger **`t`** den aktuellen Knoten, den Sie untersuchen, und in **`prev`** den Vorgängerknoten.
(3) Enden wir in der Situation, dass wir die gesamte Liste durchlaufen haben bis zum letzten **`NULL`**, dann kam der Wert nicht vor.
(4) Andernfalls muss man nur den Pointer **`prev->next`** umlenken, um den Knoten zu löschen, und den Speicherplatz des Hilfszeigers **`t`** freigeben.

Lösung:

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

typedef struct Knoten_ {
 int data;
  struct Knoten_ *next;
  } Knoten;

  Knoten *kopf = NULL;
int anzahl = 0;


 void einfuegenAmAnfang(int x) {
 Knoten *t;

 t = (Knoten *)malloc(sizeof(Knoten));
 anzahl++;
 if (kopf == NULL) {
 kopf = t;
 kopf->data = x;
 kopf->next = NULL;
 }
 else {
 t->data = x;
 t->next = kopf;
 kopf = t;
 }
 }

 void einfuegenAmEnde(int x) {
 Knoten *t, *temp;

 t = (Knoten*)malloc(sizeof(Knoten));
 anzahl++;

 if (kopf == NULL) {
 kopf = t;
 kopf->data = x;
 kopf->next = NULL;
 }
 else {
 temp = kopf;

 while (temp->next != NULL)
 temp = temp->next;

 temp->next = t;
```

```
t->data = x;
t->next = NULL;
}
}

void ausgeben() {
Knoten *t;

t = kopf;

if (t == NULL) {
printf("Die Linked List ist leer.\n");
}
else {
printf("Die Linked List hat %d Elemente.\n", anzahl);


while (t->next != NULL) {
printf("%d ", t->data);
t = t->next;
}
printf("%d\n", t->data);
}
}

void loeschenAmAnfang() {
 Knoten *t;
 int n;

 if (kopf == NULL) {
   printf("Die Linked List ist schon leer.\n");
 }
 else {
   n = kopf->data;
   t = kopf->next;
   free(kopf);
   kopf = t;
   anzahl--;
   printf("%d wurde vom Anfang geloescht.\n", n);
 }
}

void loescheKnoten(int x){
 Knoten *t, *prev;
 t = kopf;
 int n,flag=1;
 if(t == NULL){
   printf("Die Linked List ist schon leer.\n");
 }
 else{
   if(t->data == x){
     loeschenAmAnfang();
     flag=0;
   }
   else{
     while(t->next != NULL && t->next->data != x && flag == 1){
     t=t->next;
     }
     if(t->next == NULL){
     printf("%i ist nicht drin\n",x);
     }
       prev=t->next;// ich habe "prev" verwendet um zu Knot am Ende lösche
       n=prev->data;// der Zahl ist data von gelöschene Knot
       t->next=t->next->next;
       free(prev);
       anzahl--;
       printf("%i wurde geloescht",n);
   }

 }
}




void loeschenAmEnde() {
 Knoten *t, *u;
 int n;

if (kopf == NULL) {
```

```
  printf("Die Linked List ist schon leer.\n");
 }
  else {
    anzahl--;

    if (kopf->next == NULL) {
      n = kopf->data;
      free(kopf);
      kopf = NULL;
      printf("%d wurde vom Ende geloescht.\n", n);
    }
    else {
      t = kopf;

      while (t->next != NULL) {
      u = t;
      t = t->next;
      }

      n = t->data;
      u->next = NULL;
      free(t);
      printf("%d wurde vom Ende geloescht.\n", n);
    }
  }
}

int main() {
int auswahl, data;
int weiter = 1;

while (weiter) {
printf("\nWas wollen Sie mit der Linked List tun?\n\n");
printf("1. Einfuegen am Anfang\n");
printf("2. Einfuegen am Ende\n");
printf("3. Ausgabe der Linked List\n");
printf("4. Loeschen am Anfang\n");
printf("5. Loeschen am Ende\n");
printf("6. Ende\n");
printf("7. Löschen\n");
printf("Auswahl: ");
scanf("%d", &auswahl);
printf("\n");

switch (auswahl) {
case 1:
printf("Wert eingeben: ");
scanf("%d", &data);
einfuegenAmAnfang(data);
break;
case 2:
printf("Wert eingeben: ");
scanf("%d", &data);
einfuegenAmEnde(data);
break;
case 3:
ausgeben();
break;
case 4:
loeschenAmAnfang();
break;
case 5:
loeschenAmEnde();
break;
case 6:
weiter = 0;
break;
case 7:
printf("wert eingeben: ");
scanf("%d",&data);
loescheKnoten(data);
break;
default:
printf("Bitte einen sinnvollen Wert eingeben. \n");
}
}
return 0;
}
```

Aufgabe 3: Programmieren in C

Schreiben Sie eine **rekursive Funktion** für die *Fibonacci-Zahlen*:

$f_n = f_{n-1} + f_{n-2}$ für n >=2 und
$f_0 = 0$ und $f_1 = 1$, sonst.

Schreiben Sie auch eine **`main`**-Funktion, die eine Zahl von der Tastatur einliest und dann das Fibonacci-Ergebnis für diese Zahl ausgibt. Bitte testen Sie, ab wann der Berechnungsaufwand zu groß wird, so dass Sie nicht auf das Ergebnis warten können.

Lösung:

```
#include <stdio.h


int fibbonacci(int n) {
   if(n == 1){
      return 1;
   } else if(n == 0) {
      return 0;
   } else {
      return (fibbonacci(n-1) + fibbonacci(n-2));
   }
}

int main()
{
    int n;
    printf("Bitte geben ein Zahl ein: ");
    scanf("%i",&n);
    printf("fibbonacci Ergebnis = %i\n", fibbonacci(n));
    printf("fibbonacci Folge = ");
    for(int i=0;i<=n;i++){
      printf("%i  ",fibbonacci(i));
    }
    return 0;
}
```

Aufgabe 4: Programmieren in C

Schreiben Sie eine **iterative Funktion** für die *Fibonacci-Zahlen* und eine **`main`**-Funktion, die eine Zahl von der Tastatur einliest und dann das Fibonacci-Ergebnis für diese Zahl ausgibt. Benutzen Sie hierfür ein **Array**, das die Fibonacci-Zahlen enthält.

Lösung:

```
#include <stdio.h>


void fibbonacci(int n) {
  int a[100];
  printf("fibbonacci Folge = ");
  for(int i=0;i<=n;i++){
    if(i == 0)
```

```
      a[i] = 0;
    else if(i == 1)
      a[i] = 1;
    else if(i == 2)
      a[i] = 1;
    else{
      a[i] = a[i-2] + a[i-1];
    }
    printf("%i  ",a[i]);

  }
  printf("\nfibbonacci Ergebnis = %i ",a[n]);
}
int main()
{
    int n;
    printf("Bitte geben ein Zahl ein: ");
    scanf("%i",&n);
    fibbonacci(n);

    return 0;
}
```